ALIBABA CLOUD

阿里云

资源编排

资源编排公共云合集

文档版本：20220610

阿里云

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。

2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。

3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。

4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。

5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。

6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

格式	说明	样例
危险	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	危险 重置操作将丢失用户配置数据。
警告	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	警告 重启操作将导致业务中断，恢复业务时间约十分钟。
注意	用于警示信息、补充说明等，是用户必须了解的内容。	注意 权重设置为0，该服务器不会再接受新请求。
说明	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	说明 您也可以通过按Ctrl+A选中全部文件。
>	多级菜单递进。	单击**设置**> **网络**> **设置网络类型**。
粗体	表示按键、菜单、页面名称等UI元素。	在**结果确认**页面，单击**确定**。
Courier字体	命令或代码。	执行 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *Instance_ID*
[] 或者 [a|b]	表示可选项，至多选择一个。	`ipconfig [-all|-t]`
{} 或者 {a|b}	表示必选项，至多选择一个。	`switch {active|stand}`

目录

1.动态与公告

1.1. 新功能发布记录

本文介绍了资源编排服务ROS（Resource Orchestration Service）产品主要功能的发布时间和相关文档。

2022年04月

功能名称	功能描述	相关文档
云服务ECS、VPC、SLB、Kafka、PolarDB、MongoDB和ElasticSearch的资源支持自动绑定系统标签	以下资源类型支持自动绑定系统标签： • ALIYUN::ECS::AutoSnapshotPolicy • ALIYUN::ECS::SecurityGroupClone • ALIYUN::ECS::VPC • ALIYUN::ECS::VSwitch • ALIYUN::VPC::RouteTable • ALIYUN::VPC::EIP • ALIYUN::VPC::EIPPro • ALIYUN::VPC::CommonBandwidthPackage • ALIYUN::VPC::VpnGateway • ALIYUN::SLB::AccessControl • ALIYUN::SLB::Certificate • ALIYUN::KAFKA::Instance • ALIYUN::KAFKA::Topic • ALIYUN::POLARDB::DBCluster • ALIYUN::REDIS::Instance • ALIYUN::MONGODB::Instance • ALIYUN::MONGODB::ShardingInstance • ALIYUN::MONGODB::ServerlessInstance • ALIYUN::ElasticSearch::Instance	• ALIYUN::ECS::AutoSnapshotPolicy • ALIYUN::ECS::SecurityGroupClone • ALIYUN::ECS::VPC • ALIYUN::ECS::VSwitch • ALIYUN::VPC::RouteTable • ALIYUN::VPC::EIP • ALIYUN::VPC::EIPPro • ALIYUN::VPC::CommonBandwidthPackage • ALIYUN::VPC::VpnGateway • ALIYUN::SLB::AccessControl • ALIYUN::SLB::Certificate • ALIYUN::KAFKA::Instance • ALIYUN::KAFKA::Topic • ALIYUN::POLARDB::DBCluster • ALIYUN::REDIS::Instance • ALIYUN::MONGODB::Instance • ALIYUN::MONGODB::ShardingInstance • ALIYUN::MONGODB::ServerlessInstance • ALIYUN::ElasticSearch::Instance
云服务ECS支持资源清理	以下资源类型支持资源清理： • ALIYUN::ECS::DedicatedHost • ALIYUN::ECS::HpcCluster • ALIYUN::ECS::SSHKeyPair • ALIYUN::ECS::DeploymentSet • ALIYUN::ECS::LaunchTemplate	• ALIYUN::ECS::DedicatedHost • ALIYUN::ECS::HpcCluster • ALIYUN::ECS::SSHKeyPair • ALIYUN::ECS::DeploymentSet • ALIYUN::ECS::LaunchTemplate

功能名称	功能描述	相关文档
Fn::Sub和Fn::Calculate函数能力增强	• Fn::Sub：支持处理JSON对象。 • Fn::Calculate：支持取余。	• Fn::Sub • Fn::Calculate
云服务ECS新增资源类型	ALIYUN::ECS::PrefixList：创建前缀列表。	ALIYUN::ECS::PrefixList
云服务ADB、OOS、NAS、CDDC、RocketMQ、PVTZ、FC、CS的资源支持更新标签属性	以下云服务的资源支持更新资源组ID属性： • ALIYUN::ADB::DBCluster • ALIYUN::OOS::Template • ALIYUN::OOS::Execution • ALIYUN::NAS::FileSystem • ALIYUN::CDDC::DedicatedHost • ALIYUN::ROCKETMQ::Instance • ALIYUN::PVTZ::Zone • ALIYUN::FC::Service • ALIYUN::CS::AnyCluster • ALIYUN::CS::KubernetesCluster • ALIYUN::CS::ManagedEdgeKubernetesCluster • ALIYUN::CS::ManagedKubernetesCluster • ALIYUN::CS::ServerlessKubernetesCluster	• ALIYUN::ADB::DBCluster • ALIYUN::OOS::Template • ALIYUN::OOS::Execution • ALIYUN::NAS::FileSystem • ALIYUN::CDDC::DedicatedHost • ALIYUN::ROCKETMQ::Instance • ALIYUN::PVTZ::Zone • ALIYUN::FC::Service • ALIYUN::CS::AnyCluster • ALIYUN::CS::KubernetesCluster • ALIYUN::CS::ManagedEdgeKubernetesCluster • ALIYUN::CS::ManagedKubernetesCluster • ALIYUN::CS::ServerlessKubernetesCluster
GetFeatureDetails接口支持查询资源场景支持的资源类型	GetFeatureDetails接口支持指定Feature=TemplateScratch，返回资源场景支持的资源类型。	GetFeatureDetails
资源场景支持PolarDB	资源场景支持对PolarDB集群进行复制、导入或迁移。	ALIYUN::POLARDB::DBCluster
容器服务和ECS支持参数查询功能	支持容器节点按地域维度查询，支持ECS多可用区查询。	DATASOURCE::ECS::RecommendInstanceTypes

功能名称	功能描述	相关文档
云服务ALB、DNS、ECS、Redis、PVTZ、RDS和SLB的资源支持更新标签属性	以下云服务的资源支持更新资源组ID属性： • ALIYUN::ALB::LoadBalancer • ALIYUN::ALB::ServerGroup • ALIYUN::DNS::Domain • ALIYUN::ECS::VPC • ALIYUN::ECS::AutoSnapshotPolicy • ALIYUN::ECS::CopyImage • ALIYUN::ECS::CustomImage • ALIYUN::ECS::Disk • ALIYUN::ECS::NetworkInterface • ALIYUN::ECS::DedicatedHost • ALIYUN::ECS::Instance • ALIYUN::ECS::InstanceGroup • ALIYUN::ECS::SecurityGroup • ALIYUN::ECS::Snapshot • ALIYUN::ECS::SSHKeyPair • ALIYUN::REDIS::Instance • ALIYUN::PVTZ::Zone • ALIYUN::RDS::DBInstance • ALIYUN::SLB::AccessControl • ALIYUN::SLB::Certificate • ALIYUN::SLB::LoadBalancer • ALIYUN::VPC::CommonBandwidthPackage • ALIYUN::VPC::EIP、ALIYUN::VPC::EIPPro	• ALIYUN::ALB::LoadBalancer • ALIYUN::ALB::ServerGroup • ALIYUN::DNS::Domain • ALIYUN::ECS::VPC • ALIYUN::ECS::AutoSnapshotPolicy • ALIYUN::ECS::CopyImage • ALIYUN::ECS::CustomImage • ALIYUN::ECS::Disk • ALIYUN::ECS::NetworkInterface • ALIYUN::ECS::DedicatedHost • ALIYUN::ECS::Instance • ALIYUN::ECS::InstanceGroup • ALIYUN::ECS::SecurityGroup • ALIYUN::ECS::Snapshot • ALIYUN::ECS::SSHKeyPair • ALIYUN::REDIS::Instance • ALIYUN::PVTZ::Zone • ALIYUN::RDS::DBInstance • ALIYUN::SLB::AccessControl • ALIYUN::SLB::LoadBalancer • ALIYUN::VPC::CommonBandwidthPackage • ALIYUN::VPC::EIP • ALIYUN::VPC::EIPPro

功能名称	功能描述	相关文档
云服务FC、SLS、CR、CS、SAE、MNS和OTS的资源支持数据源	• DATASOURCE::FC::Services：查询函数计算服务的服务列表。 • DATASOURCE::FC::Functions：查询函数计算服务的函数列表。 • DATASOURCE::FC::Aliases：查询函数计算服务的别名列表。 • DATASOURCE::FC::Triggers：查询函数计算服务的触发器列表。 • DATASOURCE::FC::CustomDomains：查询函数计算服务的自定义域名列表。 • DATASOURCE::SLS::Project：查询日志服务的项目列表。 • DATASOURCE::SLS::Logstores：查询日志服务的日志库列表。 • DATASOURCE::CR::Namespaces：查询容器镜像服务的命名空间列表。 • DATASOURCE::CR::Repositories：查询容器镜像服务的仓库列表。 • DATASOURCE::CS::KubernetesClusters：查询容器服务的集群列表。 • DATASOURCE::CS::ClusterNodePools：查询容器服务的集群节点池列表。 • DATASOURCE::SAE::Applications：查询Serverless应用引擎服务的应用列表。 • DATASOURCE::SAE::Namespace：查询Serverless应用引擎服务的命名空间列表。 • DATASOURCE::MNS::Queues：查询Se消息服务MNS的队列列表。 • DATASOURCE::MNS::Topics：查询Se消息服务MNS的主题列表。 • DATASOURCE::MNS::Subscriptions：查询Se消息服务MNS的订阅列表。 • DATASOURCE::OTS::Instances：查询表格存储服务的实例列表。	• DATASOURCE::FC::Services • DATASOURCE::FC::Functions • DATASOURCE::FC::Aliases • DATASOURCE::FC::Triggers • DATASOURCE::FC::CustomDomains • ALIYUN::SLS::Project • DATASOURCE::SLS::Logstores • DATASOURCE::CR::Namespaces • DATASOURCE::CR::Repositories • ALIYUN::CS::KubernetesCluster • ALIYUN::CS::ClusterNodePool • DATASOURCE::SAE::Applications • ALIYUN::SAE::Namespace • DATASOURCE::MNS::Queues • DATASOURCE::MNS::Topics • DATASOURCE::MNS::Subscriptions • DATASOURCE::OTS::Instances
参数查询功能支持对容器服务相关资源的ZoneIds进行查询	支持对容器服务相关资源的ZoneIds进行查询。	无

2022年03月

功能名称	功能描述	相关文档
校验和创建模板支持Count和DependsOn表达式	使用ValidateTemplate和CreateTemplate接口校验和创建模板时，支持模板中使用Count和DependsOn表达式。	• Count • DependsOn
云服务RAM和NAS新增数据源	• DATASOURCE::RAM::Users：查询RAM用户列表。 • DATASOURCE::RAM::Groups：查询RAM用户组列表。 • DATASOURCE::RAM::Roles：查询RAM角色列表。 • DATASOURCE::RAM::Policies：查询RAM策略列表。 • DATASOURCE::NAS::FileSystems：查询NAS文件系统列表。 • DATASOURCE::NAS::Zones：查询NAS可用区列表。	• DATASOURCE::RAM::Users • DATASOURCE::RAM::Groups • DATASOURCE::RAM::Roles • DATASOURCE::RAM::Policies • DATASOURCE::NAS::FileSystems • DATASOURCE::NAS::Zones
资源场景支持RAM相关资源，支持禁用名称唯一性	资源场景支持对以下资源类型进行复制、导入或迁移： • ALIYUN::RAM::User • ALIYUN::RAM::Group • ALIYUN::RAM::Role • ALIYUN::RAM::ManagedPolicy 创建资源场景时PreferenceParameters支持指定DisableNameUnique，用来禁用名称唯一性保障。	• ALIYUN::RAM::User • ALIYUN::RAM::Group • ALIYUN::RAM::Role • ALIYUN::RAM::ManagedPolicy
云服务ECS新增数据源	• DATASOURCE::ECS::AutoSnapshotPolicies：查询自动快照策略列表。 • DATASOURCE::ECS::Snapshots：查询快照列表。 • DATASOURCE::ECS::DedicatedHosts：查询专有宿主机列表。 • DATASOURCE::ECS::DeploymentSets：查询部署集列表。 • DATASOURCE::ECS::HpcClusters：查询HPC集群列表。 • DATASOURCE::ECS::KeyPairs：查询SSH密钥对列表。	• DATASOURCE::ECS::AutoSnapshotPolicies • DATASOURCE::ECS::Snapshots • DATASOURCE::ECS::DedicatedHosts • DATASOURCE::ECS::DeploymentSets • DATASOURCE::ECS::HpcClusters • DATASOURCE::ECS::KeyPairs
云服务VPC新增资源	ALIYUN::VPC::EIPPro：创建指定的弹性公网IP。	ALIYUN::VPC::EIPPro

功能名称	功能描述	相关文档
云服务RDS的资源支持资源清理	ALIYUN::RDS:DBInstance：支持资源清理。	ALIYUN::RDS::DBInstance
云服务RDS和ElasticSearch的资源支持参数查询功能	支持ElasticSearch实例规格查询，支持对RDS实例规格查询性能优化。	• ALIYUN::ElasticSearch::Instance • ALIYUN::RDS::DBInstance
Terraform参数提取规则调整	any类型在无默认值时推断为String。	Parameters（可选）
云服务VPC新的数据源	• DATASOURCE::VPC::NatGatways：查询NAT网关列表。 • DATASOURCE::VPC::NetworkAcls：查询网络ACL列表。	• DATASOURCE::VPC::NatGateways • DATASOURCE::VPC::NetworkAcls
云服务ALB、VPC、CDDC、ECI、ENS和SAG的资源支持询价	以下资源类型支持询价： • ALIYUN::ALB::LoadBalancer • ALIYUN::VPC::AnycastEIP • ALIYUN::CDDC::DedicatedHost • ALIYUN::ECI::ImageCache • ALIYUN::ENS::Instance • ALIYUN::SAG::SmartAccessGateway	• ALIYUN::ALB::LoadBalancer • ALIYUN::VPC::AnycastEIP • ALIYUN::CDDC::DedicatedHost • ALIYUN::ECI::ImageCache • ALIYUN::ENS::Instance • ALIYUN::SAG::SmartAccessGateway

2022年02月

功能名称	功能描述	相关文档
云服务SAG和EDAS新增资源类型	• ALIYUN::SAG::SerialNumberBinding：将智能接入网关硬件设备绑定到智能接入网关实例中。 • ALIYUN::EDAS::K8sSlbBinding：将SLB实例绑定到容器服务Kubernetes集群的应用中。	• ALIYUN::SAG::SerialNumberBinding • ALIYUN::EDAS::K8sSlbBinding
资源栈支持托管模式	资源栈托管模式支持解决云服务基于ROS的权限控制问题。由云产品扮演用户角色创建，对用户可见但不能被修改。	GetStack

功能名称	功能描述	相关文档
云服务SAG新增资源类型	• ALIYUN::SAG::Qos：创建QoS策略实例。 • ALIYUN::SAG::QosAssociation：将QoS策略应用到智能接入网关实例。 • ALIYUN::SAG::QosCar：创建QoS的限速规则。 • ALIYUN::SAG::QosPolicy：创建QoS策略流分类规则。	• ALIYUN::SAG::Qos • ALIYUN::SAG::QosAssociation • ALIYUN::SAG::QosCar • ALIYUN::SAG::QosPolicy
GetServiceProvisions接口新增查询EHPC服务的开通状态和服务角色信息	支持查询EHPC服务的开通状态和服务角色信息。	GetServiceProvisions

2022年01月

功能名称	功能描述	相关文档
云服务ALB新增资源类型	• ALIYUN::ALB::BackendServerAttachment：向服务器组中添加后端服务器。 • ALIYUN::ALB::Listener：创建HTTP、HTTPS或QUIC监听。 • ALIYUN::ALB::Rule：创建转发规则。 • ALIYUN::ALB::ServerGroup：创建服务器组。	• ALIYUN::ALB::BackendServerAttachment • ALIYUN::ALB::Listener • ALIYUN::ALB::Rule • ALIYUN::ALB::ServerGroup
支持Terraform 1.1版本	ROS模板的Transform属性值支持使用Terraform 1.1.x版本。	Transform（必选）
新增资源清理器	ALIYUN::ROS::ResourceCleaner：进行资源清理。	ALIYUN::ROS::ResourceCleaner
新增数据源资源功能，支持查询云服务的资源数据	支持查询云服务的资源数据。数据源资源可以被其他资源引用，也可以在输出中被引用。	数据源资源
云服务ECS新增数据源	• DATASOURCE::ECS::RecommendInstanceTypes：查询ECS实例推荐规格列表。 • DATASOURCE::ECS::Images：查询ECS镜像列表。 • DATASOURCE::ECS::DiskCategories：查询ECS磁盘类型列表。	• DATASOURCE::ECS::RecommendInstanceTypes • DATASOURCE::ECS::Images • DATASOURCE::ECS::DiskCategories

功能名称	功能描述	相关文档
云服务TSDB和SAG新增资源类型	• ALIYUN::TSDB::InfluxDBDatabase：创建时间序列数据库。 • ALIYUN::TSDB::InfluxDBUser：创建时间序列数据库账号。 • ALIYUN::SAG::SmartAccessGateway：创建智能接入网关实例。	• ALIYUN::TSDB::InfluxDBDatabase • ALIYUN::TSDB::InfluxDBUser • ALIYUN::SAG::SmartAccessGateway
云服务ECS的资源支持资源清理	ALIYUN::ECS::Instance：进行资源清理。	ALIYUN::ECS::Instance
云服务Elasticsearch和EMR的资源支持询价、系统标签、用户标签和资源组	• alicloud_elasticsearch_instance：支持询价、系统标签、用户标签、资源组。 • alicloud_emr_cluster：支持询价、用户标签、资源组。	ROS资源支持情况
嵌套资源栈和Fn::GetStackOutput支持Terrraform模板	• 嵌套资源栈支持使用Terraform模板。 • Fn::GetStackOutput支持获取基于Terraform模板创建的资源栈的输出。	Terraform类型模板结构
云服务GPDB的资源支持参数查询功能	支持对以下资源的可用区查询： • ALIYUN::GPDB::DBInstance • ALIYUN::GPDB::ElasticDBInstance	• ALIYUN::GPDB::DBInstance • ALIYUN::GPDB::ElasticDBInstance

2021年12月

功能名称	功能描述	相关文档
Terraform支持资源组传递功能	Terraform支持资源组传递功能。	ROS功能支持情况
Terraform的ECS和VPC资源支持标签	alicloud_key_pair、alicloud_security_group、alicloud_vpc、alicloud_vswitch支持用户标签和系统标签。	ROS资源支持情况
更改集和资源栈接口支持资源场景	CreateChangeSet支持资源迁移类型的资源场景。CreateStack、PreviewStack、GetTemplateEstimateCost支持指定TemplateScratchRegionId，以支持跨地域迁移和复制。	• CreateChangeSet • CreateStack • PreviewStack • GetTemplateEstimateCost

功能名称	功能描述	相关文档
支持将模板共享给资源目录中的成员	在资源目录内，支持使用阿里云提供的资源共享功能。管理员或成员可以将自己的ROS模板共享给其他成员使用，以便企业在特定的账号下统一管理模板，降低操作成本。	将模板共享给资源目录中的成员
云服务ALB和ROS新增资源类型	• ALIYUN::ALB::LoadBalancere：创建应用型负载均衡ALB实例。 • ALIYUN::ROS::Sleep：用于延迟其他资源的创建、删除、更新和回滚等操作。	• ALIYUN::ALB::LoadBalancer • ALIYUN::ROS::Sleep
Terraform的ECI、MarketPlace、RDS和ECS的资源支持询价、标签和所属资源组	• 支持询价：alicloud_market_order、alicloud_eci_container_group、alicloud_drds_instance。 • 支持用户标签：alicloud_snapshot、alicloud_image_copy、alicloud_image、alicloud_ecs_network_interface、alicloud_ecs_launch_template、alicloud_drds_instance。 • 支持系统标签：alicloud_snapshot、alicloud_image_copy、alicloud_image、alicloud_ecs_network_interface、alicloud_ecs_launch_template。 • 支持资源组：alicloud_image_copy、alicloud_image、alicloud_ecs_network_interface、alicloud_ecs_launch_template、alicloud_drds_instance、alicloud_eci_container_group。	ROS资源支持情况

2021年11月

功能名称	功能描述	相关文档
Terraform支持指定Provider版本	Terraform新增支持指定Provider版本。目前，在使用Terraform 1.0.10时，可指定Provider为alicloud的版本范围为：1.139.0~1.140.0。	Terraform支持的功能和资源

功能名称	功能描述	相关文档
Terraform支持新的Provider	Terraform新增支持Provider helm和kubernetes。	Terraform支持的功能和资源
支持服务查询接口	新增支持GetServiceProvisions接口，用于查询服务开通状态和服务角色信息。	GetServiceProvisions
Terraform支持更改集和资源栈组	Terraform资源栈支持更改集功能。此外，支持使用Terraform类型的模板创建资源栈组。	数据结构
支持资源场景功能	资源场景功能支持在可视化界面上选择资源范围，并对一组资源进行复制、纳管和迁移等操作，从而简化资源管理。	概览
Terraform支持询价和标签	• 支持询价、用户标签：alicloud_cen_bandwidth_package、alicloud_mongodb_instance、alicloud_polardb_cluster。 • 支持系统标签：alicloud_polardb_cluster。	ROS资源支持情况

2021年10月

功能名称	功能描述	相关文档
Terraform支持询价	ROS控制台和GetTemplateEstimateCost接口支持对Terraform模板中的资源进行询价。	• Terraform支持的功能和资源 • GetTemplateEstimateCost

2021年09月

功能名称	功能描述	相关文档
支持云企业网、云解析PrivateZone、云原生数据仓库ADB MySQL版、容器服务Kubernetes版ACK、容器镜像服务ACR、混合云备份服务等云服务相关的资源	新增支持以下资源： • ALIYUN::CEN::CenRouteService：设置访问云服务。 • ALIYUN::PVTZ::UserVpcAuthorization：用于跨账号VPC授权。 • ALIYUN::ADB::DBCluster：创建一个AnalyticDB MySQL版集群。 • ALIYUN::CS::ClusterNodePool：为集群创建节点池。 • ALIYUN::CR::UserInfo：创建用户信息。 • ALIYUN::HBR::DbAgent：安装数据库备份客户端。	• ALIYUN::CEN::CenRouteService • ALIYUN::PVTZ::UserVpcAuthorization • ALIYUN::ADB::DBCluster • ALIYUN::CS::ClusterNodePool • ALIYUN::CR::UserInfo • ALIYUN::HBR::DbAgent
支持Fn::MarketplaceImage函数	新增支持Fn::MarketplaceImage函数，用于返回指定云市场镜像商品Code的默认镜像ID。	函数（Functions）

功能名称	功能描述	相关文档
Terraform支持并发控制	CreateStack、UpdateStack、PreviewStack、ContinueCreateStack接口支持通过指定Parallelism参数为Terraform资源栈设定并发数。	• CreateStack • UpdateStack • PreviewStack • ContinueCreateStack
新增模板参数取值查询接口	新增支持GetTemplateParameterConstraints接口，用于查询模板参数的取值。	GetTemplateParameterConstraints

2021年08月

功能名称	功能描述	相关文档
支持专有网络相关资源	新增支持以下资源： • ALIYUN::VPC::DhcpOptionsSet：创建动态主机配置协议DHCP选项集。 • ALIYUN::VPC::DhcpOptionsSetAttachment：将动态主机配置协议DHCP选项集关联到专有网络VPC。	• ALIYUN::VPC::DhcpOptionsSet • ALIYUN::VPC::DhcpOptionsSetAttachment
Terraform支持新的Provider	Terraform新增支持Provider fortios和fortimanager。	Terraform支持的功能和资源
支持函数计算相关资源	新增支持ALIYUN::FC::Layer，表示发布层版本。	ALIYUN::FC::Layer

资源栈组支持服务管理权限模式，可以将资源目录的资源夹作为部署资源栈的为目标。	支持创建自助管理权限模式或服务管理权限模式的资源栈组，具体如下：
	• 资源栈组原有的权限模式为自助管理权限模式，当创建自助管理权限模式的资源栈组时，需要事先在管理员账号和目标账号中手动创建RAM角色，建立二者的信任关系，然后在其他阿里云账号中部署资源栈。
	• 新增服务管理权限模式，当创建服务管理权限模式的资源栈组时，只需开启可信访问，ROS会为管理员账号和目标账号自动创建服务关联角色，管理员账号通过服务关联角色在目标账号中部署资源栈。服务管理权限模式优势如下：
	  ○ 用户无需事先手动创建资源栈组所需的RAM角色，ROS会根据情况自动创建并管理RAM角色。
	  ○ 资源栈组支持以资源目录的资源夹为目标进行部署。同时支持监听资源夹中的成员账号变动，以进行自动部署或删除资源栈实例。
		概览

ValidateTemplate接口支持指定校验选项	ValidateTemplate接口新增支持参数ValidationOption，用于指定校验选项。取值： • None（默认值）：不开启额外校验。 • EnableTerraformValidation：对于Terraform类型模板，使用Terraform CLI的validate命令开启额外校验。	ValidateTemplate

2021年07月

功能名称	功能描述	相关文档
API网关、函数计算、日志服务、Serverless应用引擎、对象存储服务、容器服务Kubernetes版ACK、专有网络、文件存储、云服务器等云服务的资源支持更新标签属性	以下资源支持更新标签属性： • ALIYUN::ApiGateway::Group：创建API分组。 • ALIYUN::ApiGateway::App：创建应用。 • ALIYUN::FC::Service：创建函数计算服务。 • ALIYUN::SLS::Project：创建日志项目。 • ALIYUN::SAE::Application：创建轻量级分布式应用服务应用。 • ALIYUN::OSS::Bucket：创建OSS存储空间。 • ALIYUN::CS::AnyCluster：创建任意类型的Kubernetes集群实例。 • ALIYUN::CS::KubernetesCluster：创建Kubernetes专有版集群。 • ALIYUN::CS::ManagedEdgeKubernetesCluster：创建Kubernetes边缘托管版集群实例。 • ALIYUN::CS::ManagedKubernetesCluster：创建Kubernetes托管版集群。 • ALIYUN::CS::ServerlessKubernetesCluster：创建Serverless Kubernetes集群实例。 • ALIYUN::VPC::EIP：申请弹性公网IP。 • ALIYUN::NAS::FileSystem：创建新的文件系统。 • ALIYUN::ECS::SecurityGroup：创建安全组。	• ALIYUN::ApiGateway::Group • ALIYUN::ApiGateway::App • ALIYUN::FC::Service • ALIYUN::SLS::Project • ALIYUN::SAE::Application • ALIYUN::OSS::Bucket • ALIYUN::CS::AnyCluster • ALIYUN::CS::KubernetesCluster • ALIYUN::CS::ManagedEdgeKubernetesCluster • ALIYUN::CS::ManagedKubernetesCluster • ALIYUN::CS::ServerlessKubernetesCluster • ALIYUN::VPC::EIP • ALIYUN::NAS::FileSystem • ALIYUN::ECS::SecurityGroup
支持日志服务相关资源	新增支持资源ALIYUN::SLS::Etl，表示创建数据加工任务。	ALIYUN::SLS::Etl
支持查询资源创建进度	• ALIYUN::ROS::WaitCondition资源新增支持ShowProgressEvent属性，用于控制是否显示接收UserData消息的进度事件。 • GetStack接口新增ShowResourceProgress参数，用于控制是否返回资源处理进度。	• ALIYUN::ROS::WaitCondition • GetStack

Terraform支持新的功能和版本	• Terraform支持参数、继续创建功能。
	• 支持指定角色或使用STS方式创建Terraform。
	• 资源栈新增支持Terraform 1.0版本。
	• 新增支持3个Provider：
	  ○ time：时间相关功能。
	  ○ random：随机数相关功能。
	  ○ template：模板相关功能。
		Terraform支持的功能和资源

功能名称	功能描述	相关文档
Terraform支持伪参数功能	Terraform模板中新增支持以下伪参数： • ALIYUN__StackId：对应 ALIYUN::StackId。 • ALIYUN__StackName：对应ALIYUN::StackName。 • ALIYUN__TenantId：对应ALIYUN::TenantId。 • ALIYUN__Region：对应ALIYUN::Region。 • ALIYUN__AccountId：对应ALIYUN::AccountId。 • ALIYUN__NoValue：对应ALIYUN::NoValue。	Terraform类型模板结构
支持云原生数据仓库AnalyticDB PostgreSQL版相关资源	新增支持以下资源： • ALIYUN::GPDB::DBInstance：创建存储预留模式的AnalyticDB for PostgreSQL实例。 • ALIYUN::GPDB::ElasticDBInstance：创建存储弹性模式的AnalyticDB for PostgreSQL实例。 • ALIYUN::GPDB::InstancePublicConnection：分配实例外网连接地址。	• ALIYUN::GPDB::DBInstance • ALIYUN::GPDB::ElasticDBInstance • ALIYUN::GPDB::InstancePublicConnection

2021年06月

功能名称	功能描述	相关文档
支持Fn::Any函数	新增支持Fn::Any函数，用于返回指定数组中取值的真假情况。当数组中任意一项为真时，返回true（真），否则返回false（假）。	函数（Functions）
支持资源管理、大数据计算服务、阿里云E-MapReduce等云服务相关的资源	新增支持以下资源： • ALIYUN::ResourceManager::ResourceShare：创建共享单元。 • ALIYUN::MaxCompute::Table：创建表。 • ALIYUN::EMR::ClusterServiceConfigs：创建或修改集群指定服务的配置信息。	• ALIYUN::ResourceManager::ResourceShare • ALIYUN::MaxCompute::Table • ALIYUN::EMR::ClusterServiceConfigs

2021年05月

功能名称	功能描述	相关文档
支持自动开通新的服务	ALIYUN::ROS::AutoEnableService支持自动开通企业级分布式应用服务EDAS。	ALIYUN::ROS::AutoEnableService

功能名称	功能描述	相关文档
支持云防火墙、专有网络等云服务相关的资源	新增支持以下资源： • ALIYUN::CLOUDFW::VpcFirewallControlPolicy：为指定VPC边界防火墙策略组添加访问控制策略。 • ALIYUN::VPC::FlowLog：创建流日志。 • ALIYUN::VPC::VpnRouteEntry：创建VPN目的路由。	• ALIYUN::CLOUDFW::VpcFirewallControlPolicy • ALIYUN::VPC::FlowLog • ALIYUN::VPC::VpnRouteEntry

2021年04月

功能名称	功能描述	相关文档
支持Fn::FormatTime函数	新增支持Fn::FormatTime函数，用于返回格式化后的当前时间。	函数（Functions）
支持企业级分布式应用服务、API网关、配置审计、访问控制、日志服务、Web应用防火墙等云服务相关的资源	新增支持以下资源： • ALIYUN::EDAS::K8sCluster：创建K8s集群。 • ALIYUN::EDAS::K8sApplication：在Kubernetes集群中创建应用。 • ALIYUN::ApiGateway::LogConfig：创建日志配置。 • ALIYUN::ApiGateway::PluginAttachment：将插件绑定到API。 • ALIYUN::Config::Rule：新建或修改规则。 • ALIYUN::Config::DeliveryChannel：创建或更新投递渠道。 • ALIYUN::RAM::RamAccountAlias：创建账号别名。 • ALIYUN::RAM::SecurityPreference：设置RAM用户的全局安全首选项。 • ALIYUN::SLS::Audit：配置日志审计服务。 • ALIYUN::WAF::LogServiceEnable：开启指定域名配置的日志采集功能。	• ALIYUN::EDAS::K8sCluster • ALIYUN::EDAS::K8sApplication • ALIYUN::ApiGateway::LogConfig • ALIYUN::ApiGateway::PluginAttachment • ALIYUN::Config::Rule • ALIYUN::Config::DeliveryChannel • ALIYUN::RAM::RamAccountAlias • ALIYUN::RAM::SecurityPreference • ALIYUN::SLS::Audit • ALIYUN::WAF::LogServiceEnable

2021年03月

功能名称	功能描述	相关文档
更新更改集功能	更改集新增支持模板中的条件（Conditions）和映射（Mappings）。	无

功能名称	功能描述	相关文档
支持资源组功能	• 创建资源栈、资源栈组、模板时支持指定资源组。 • 创建资源栈时指定资源组，会将资源组ID传递给资源栈中每个支持资源组的资源。 • 查询资源栈、资源栈组、模板时支持返回资源组。 • 支持基于资源组的权限管理，约束对资源栈、资源栈组、模板的操作。	• CreateStack • GetStack • CreateStackGroup • GetStackGroup • CreateTemplate • GetTemplate
支持自动开通新的服务	ALIYUN::ROS::AutoEnableService支持自动开通容器镜像服务ACR、大数据计算服务MaxCompute、消息服务MNS、数据工场DataWorks和视频监控VS。	ALIYUN::ROS::AutoEnableService

2021年02月

功能名称	功能描述	相关文档
支持自动开通新的服务	ALIYUN::ROS::AutoEnableService支持自动开通工业大脑开放平台。	ALIYUN::ROS::AutoEnableService

2021年01月

功能名称	功能描述	相关文档
资源栈支持标签传递	创建或更新资源栈时若指定标签，则会传递到栈中每个支持标签的资源中。模板中定义的资源标签的优先级高于资源栈标签。	标签传递
新增资源ALIYUN::DTS::SubscriptionInstance	该资源用于创建DTS数据订阅实例、配置订阅通道。	ALIYUN::DTS::SubscriptionInstance
支持自动开通新的服务	ALIYUN::ROS::AutoEnableService支持自动开通容器服务、全站加速和链路追踪服务。	ALIYUN::ROS::AutoEnableService
ALIYUN::FC::Function新增支持实例类型	ALIYUN::FC::Function新增支持实例类型。	ALIYUN::FC::Function
PreviewStack接口优化	PreviewStack接口针对嵌套资源栈校验逻辑进行优化。	PreviewStack

2020年12月

功能名称	功能描述	相关文档
新增支持标签的资源	新增以下支持标签的资源： • ALIYUN::NAS::FileSystem • ALIYUN::SAE::Application • ALIYUN::SLS::Project	• ALIYUN::NAS::FileSystem • ALIYUN::SAE::Application • ALIYUN::SLS::Project

功能名称	功能描述	相关文档
新增支持偏差检测的资源	新增以下支持偏差检测的资源： • ALIYUN::RDS::DBInstance • ALIYUN::SLS::Savedsearch • ALIYUN::SLS::Alert • ALIYUN::OTS::Instance • ALIYUN::OTS::Table • ALIYUN::OTS::VpcBinder • ALIYUN::VPC::NatGateway • ALIYUN::OSS::Bucket	支持偏差检测和资源导入的资源类型
资源栈状态变更支持通知EventBridge	CreateStack时指定的回调地址NotificationURLs新增取值 EventBridge，可将资源栈的状态变更通知到事件总线（EventBridge）服务。	CreateStack
新增函数Fn::Index	该函数用来获取索引。	函数（Functions）
新增资源ALIYUN::CMS::SiteMonitor	该资源用于创建站点监控的监控任务。	ALIYUN::CMS::SiteMonitor
新增资源ALIYUN::TSDB::HiTSDBInstance	该资源用于创建时序数据库实例。	ALIYUN::TSDB::HiTSDBInstance
新增资源ALIYUN::IOT::Rule	该资源用于为指定Topic创建规则。	ALIYUN::IOT::Rule
新增资源ALIYUN::IOT::RuleAction	该资源用于在指定的规则下创建一个规则动作。	ALIYUN::IOT::RuleAction
新增资源ALIYUN::IOT::ProductTopic	该资源用于为指定产品创建自定义Topic类。	ALIYUN::IOT::ProductTopic
新增资源ALIYUN::PrivateLink::VpcEndpointService	该资源用于创建私网连接的终端节点服务。	ALIYUN::PrivateLink::VpcEndpointService
新增资源ALIYUN::PrivateLink::VpcEndpoint	该资源用于创建私网连接的终端节点。	ALIYUN::PrivateLink::VpcEndpoint
新增资源ALIYUN::ASM::ServiceMesh	该资源用于创建服务网格实例。	ALIYUN::ASM::ServiceMesh
新增资源ALIYUN::ApiGateway::Instance	该资源用于创建ApiGateway专享实例。	ALIYUN::ApiGateway::Instance

功能名称	功能描述	相关文档
模板功能增强	新增模板共享功能。	SetTemplatePermission
支持自动开通新的服务	ALIYUN::ROS::AutoEnableService支持自动开通应用实时监控服务、云监控、数据总线、函数计算和私网连接服务。	ALIYUN::ROS::AutoEnableService

2020年11月

功能名称	功能描述	相关文档
旧版API支持对SourceIp的鉴权	旧API（2015-09-01）支持对SourceIp的鉴权。	无
支持OOS参数仓库	支持OOS参数仓库，您可以在ROS模板的资源和参数中使用OOS参数仓库中的参数。	使用运维编排服务参数仓库创建ROS模板中的参数
资源栈增加并发数限制	对资源栈的创建、更新、删除操作会有并发数限制，默认为 50。您可以在配额管理界面中进行调整。	使用限制
新增资源ALIYUN::KMS::Secret	该资源用于创建密钥管理服务的创建凭据，并存入凭据的初始版本。	ALIYUN::KMS::Secret
新增资源ALIYUNN::DRDS::DrdsDB	该资源用于创建PolarDB-X云原生分布式数据库。	ALIYUN::DRDS::DrdsDB
新增资源ALIYUN::OTS::SearchIndex	该资源用于在数据表上创建一个多元索引。	ALIYUN::OTS::SearchIndex

2020年10月

功能名称	功能描述	相关文档
接入配额管理平台	可在配额管理平台调整ROS的各种限制大小。	使用限制
模板参数支持选填	在模板参数中将Default设置为null，用来表示该参数选填。	概览
新增资源ALIYUN::DBS::RestoreTask	该资源用于创建数据库备份服务的恢复任务。	ALIYUN::DBS::RestoreTask
新增资源ALIYUN::HBR::RestoreJob	该资源用于创建混合云备份服务的恢复任务。	ALIYUN::HBR::RestoreJob
新增资源ALIYUN::HBR::BackupClients	该资源用于安装混合云备份服务的备份客户端。	ALIYUN::HBR::BackupClients

功能名称	功能描述	相关文档
新增支持偏差检测的资源	新增以下支持偏差检测的资源： • ALIYUN::SLB::Project • ALIYUN::SLS::Logstore • ALIYUN::SLS::Index • ALIYUN::FC::Version • ALIYUN::FC::CustomDomain • ALIYUN::FC::Alias	支持偏差检测和资源导入的资源类型
支持自动开通新的服务	ALIYUN::ROS::AutoEnableService新增自动开通智能媒体管理、密钥管理服务和消息队列RocketMQ版等服务。	ALIYUN::ROS::AutoEnableService

2020年09月

功能名称	功能描述	相关文档
模板支持新的参数	模板参数新增支持AssociationPropertyMetadata。	概览
ALIYUN::ROS::AutoEnableService支持自动开通新的服务	ALIYUN::ROS::AutoEnableService新增自动开通API网关、批量计算、移动研发平台、文件存储、混合云备份、自然语言处理、对象存储服务、表格存储和日志服务等服务。	ALIYUN::ROS::AutoEnableService
新增资源支持询价	新增以下资源支持询价： • ALIYUN::PolarDB::DBCluster • ALIYUN::ECI::ContainerGroup • ALIYUN::VPC::CommonBandwidthPackage • ALIYUN::CEN::CenBandwidthPackage • ALIYUN::ECS::DedicatedHost • ALIYUN::EMR::Cluster • ALIYUN::RDS::ReadOnlyDBInstance • ALIYUN::SAE::Application • ALIYUN::ElasticSearch::Instance • ALIYUN::Memcache::Instance • ALIYUN::EHPC::Cluster • ALIYUN::VPC::Ipv6Gateway • ALIYUN::VPC::VpnGateway • ALIYUN::DRDS::DrdsInstance	支持询价的资源类型

功能名称	功能描述	相关文档
新增支持偏差检测的资源	新增以下支持偏差检测的资源： • ALIYUN::ApiGateway::Api • ALIYUN::ApiGateway::Deployment • ALIYUN::ApiGateway::Group • ALIYUN::ApiGateway::App • ALIYUN::ApiGateway::Authoration • ALIYUN::SLS::Index • ALIYUN::FS::Service	支持偏差检测和资源导入的资源类型
新增资源ALIYUN::SLS::Alert	该资源用于创建SLS告警。	ALIYUN::SLS::Alert
新增资源ALIYUN::SLS::Savedsearch	该资源用于将SLS查询结果保存为快速查询。	ALIYUN::SLS::Savedsearch
新增资源ALIYUN::CMS::MetricRuleTemplate	该资源用于创建云监控报警规则模板。	ALIYUN::CMS::MetricRuleTemplate
新增资源ALIYUN::CMS::MonitorGroupInstances	该资源用于在云监控中添加资源到应用分组。	ALIYUN::CMS::MonitorGroupInstances
新增函数Fn::Length	该函数用于计算字符串、列表和字典的长度。	函数（Functions）

2020年08月

功能名称	功能描述	相关文档
支持Terraform的管理	控制台和API支持使用Terraform模板进行创建、更新和删除操作。	概览
控制台增加概览功能	您可以通过控制台概览页面查询所有地域的资源栈、资源栈组数量分布，以及自定义模板的数量。	无
新增支持偏差检测的资源	新增以下支持偏差检测的资源： • ALIYUN::SLS::Project • ALIYUN::SLS::Logstore • ALIYUN::REDIS::Instance • ALIYUN::REDIS::Whitelist • ALIYUN::MONGODB::Instance • ALIYUN::NAS::FileSystem • ALIYUN::SLB::MasterSlaveServerGroup • ALIYUN::VPC::SnatEntry	支持偏差检测和资源导入的资源类型

功能名称	功能描述	相关文档
新增资源ALIYUN::CMS::DynamicTagGroup	该资源用于创建云监控应用分组。	ALIYUN::CMS::DynamicTagGroup

2020年07月

功能名称	功能描述	相关文档
ContinueCreateStack接口功能优化	ContinueCreateStack接口对于嵌套资源栈相关功能进行优化。	ContinueCreateStack
新增支持偏差检测的资源	新增以下支持偏差检测的资源： • ALIYUN::ECS::Instance • ALIYUN::VPC::CommonBandwidthPackage • ALIYUN::VPC::CommonBandwidthPackageIp	支持偏差检测和资源导入的资源类型
新增资源ALIYUN::DTS::MigrationJob	该资源用于创建DTS数据迁移任务。	ALIYUN::DTS::MigrationJob
新增资源ALIYUN::RDS::Database	该资源用于创建RDS数据库。	ALIYUN::RDS::Database
新增资源ALIYUN::DMS::Instance	该资源用于创建DMS实例。	ALIYUN::DMS::Instance

2020年06月

功能名称	功能描述	相关文档
新增资源导入功能	新增支持将现有资源通过资源导入的方式，导入到ROS的资源栈中，统一进行资源管理和编排。	概览
新增资源ALIYUN::SAE::Namespace	该资源用于创建SAE命名空间。	ALIYUN::SAE::Namespace
新增资源ALIYUN::SAE::Application	该资源用于创建SAE应用。	ALIYUN::SAE::Application
新增资源ALIYUN::SAE::SlbBinding	该资源用于为SAE应用绑定SLB。	ALIYUN::SAE::SlbBinding
新增资源ALIYUN::VPC::Ipv6Gateway	该资源用于创建IPv6网关。	ALIYUN::VPC::Ipv6Gateway

功能名称	功能描述	相关文档
新增资源ALIYUN::VPC::Ipv6InternetBandwidth	该资源用于为IPv6地址购买公网带宽。	ALIYUN::VPC::Ipv6InternetBandwidth

2020年05月

功能名称	功能描述	相关文档
新增资源替换更新功能	对于不支持更新的资源属性，可以进行替换更新（即删除当前资源，再创建新资源）。	替换更新资源
新增函数Fn::GetStackOutput	该函数用于获取其他资源栈输出。	函数（Functions）
新增函数Fn::Max和Fn::Min	Fn::Max函数用于获取最大值，Fn::Min函数用于获取最小值。	函数（Functions）
新增资源ALIYUN::CMS::Contact	该资源用于创建云监控的报警联系人信息。	ALIYUN::CMS::Contact
新增资源ALIYUN::CMS::ContactGroup	该资源用于创建云监控的报警联系人组。	ALIYUN::CMS::ContactGroup
新增资源ALIYUN::CMS::MonitoringAgentProcess	该资源用于创建云监控的进程监控。	ALIYUN::CMS::MonitoringAgentProcess
新增资源ALIYUN::GWS::Cluster	该资源用于创建云桌面集群。	ALIYUN::GWS::Cluster
新增资源ALIYUN::GWS::Instance	该资源用于创建云桌面实例。	ALIYUN::GWS::Instance

2.快速入门

资源编排服务ROS（Resource Orchestration Service）是阿里云提供的一项简化云计算资源管理的服务。首次使用ROS时，您可以编写模板，定义所需的云计算资源、资源间的依赖关系等，然后创建资源栈，快速创建资源。本文以快速创建专有网络和交换机为例，为您介绍具体的操作方法。

前提条件

请确保您已经注册了阿里云账号。如还未注册，请先完成账号注册。

步骤一：编写模板

模板是一个JSON、YAML或Terraform格式的文本文件，使用UTF-8编码。您需要使用模板定义阿里云资源和配置细节，并说明资源间的依赖关系，然后基于模板创建资源栈。您可以根据模板结构和资源类型自行编写模板，也可以直接使用模板示例。

关于模板结构的更多信息，请参见JSON和YAML类型模板结构和Terraform类型模板结构。

创建专有网络和交换机的模板示例如下：

模板含义如下：

- `ROSTemplateFormatVersion`：模板的版本号，当前版本号：2015-09-01。
- `Description`：模板的描述信息，可用于说明模板的适用场景、架构说明等。通常情况下，对模板进行详细描述，有利于用户理解模板的内容。
- `Parameters`：模板的参数。示例中定义了专有网络名称（`VpcName`）、专有网络网段（`VpcCidrBlock`）、可用区ID（`ZoneId`）、交换机名称（`VSwitchName`）、交换机网段（`VSwitchCidrBlock`）和标签（`Tags`）等参数。关于参数定义的更多信息，请参见概览。
- `Resources`：模板所包含的阿里云资源。示例将创建一个专有网络和一个交换机。资源属性将引用`Parameters`中定义的参数。关于资源的更多信息，请参见ALIYUN::ECS::VPC和ALIYUN::ECS::VSwitch。
- `Outputs`：资源栈创建完成后，输出的资源信息。示例将输出专有网络ID和交换机ID。

Create a VPC and a vSwitch❯

步骤二：创建资源栈

您可以使用步骤一：编写模板编写的模板快速创建资源栈。

1. 登录资源编排控制台。
2. 在左侧导航栏，单击**资源栈**。
3. 在页面左上角的地域下拉列表，选择资源栈的所在地域，例如：华东1（杭州）。
4. 在**资源栈列表**页面，单击**创建资源栈**，然后在下拉列表中选择**使用新资源（标准）**。
5. 在**选择模板**页面，在**指定模板**区域单击**选择已有模板**、选择**模板录入方式**为**输入模板**，然后在**模板内容**区域的**ROS**页签输入步骤一：编写模板中编写的JSON格式的模板，最后单击**下一步**。
6. 在**配置模板参数**页面，输入资源栈名称，并设置以下参数。

参数	说明	示例
VpcName	专有网络名称。	myVPC

参数	说明	示例
VpcCidrBlock	专有网络网段。取值： ○ 10.0.0.0/8 ○ 172.16.0.0/12 ○ 192.168.0.0/16（默认值）	192.168.0.0/16
ZoneId	可用区ID。	华东1可用区K
VSwitchName	交换机名称。	myVSwitch
VSwitchCidrBlock	交换机网段。取值： ○ 10.0.0.0/24 ○ 172.16.0.0/24 ○ 192.168.0.0/24 **说明** 交换机跟专有网络需处于同一网段。	192.168.0.0/24
Tags	标签。 最多支持设置20个标签，每个标签由键值对组成。标签值可以为空。	[{"Key": "ros", "Value": "beginner-tutorial"}]

7. 单击**创建**。

步骤三：使用资源栈中的资源

资源栈创建成功后，您可以根据需要使用资源栈中的资源，例如：在专有网络的交换机中部署阿里云资源。

1. 在资源编排控制台的左侧导航栏，单击**资源栈**。
2. 在**资源栈列表**页面，单击目标资源栈ID。
3. 单击**资源**页签，然后单击交换机资源ID。
4. 在专有网络控制台，查看交换机基本信息，包括交换机ID、可用区、所属的专有网络ID等信息。
5. 在交换机中部署阿里云资源。

   具体操作，请参见创建云资源。

步骤四（可选）：更新资源栈

当您需要更新资源栈中的资源（例如：VpcName）时，可以更新资源栈。

1. 在资源编排控制台的左侧导航栏，单击**资源栈**。
2. 在**资源栈列表**页面，单击目标资源栈**操作**列的**更新**。
3. 在**配置模板参数**页面，更新参数信息（例如：将VpcName更新为testVPC）。
4. 单击**确认修改**。

步骤五（可选）：删除资源栈

当您不再需要已创建的资源时，可以删除资源栈，释放资源，以免产生不必要的费用。

1. 在资源编排控制台的左侧导航栏，单击**资源栈**。
2. 在**资源栈列表**页面，单击目标资源栈**操作**列的**删除**。

3. 在**删除资源栈**对话框，选择**删除方式**为**释放资源**。
4. 单击**确定**。

3.资源栈

3.1. 概览

资源栈是针对ROS资源的管理单元。您可通过创建、更新和删除资源栈来创建、更新和删除一组资源。
资源栈中的所有资源均由资源栈的模板定义。如果资源无法创建，ROS将回滚资源栈，并自动删除已创建的任何资源。如果资源无法删除，则任何剩余的资源都将保留，直到能够成功删除资源栈。

资源栈的相关内容如下表所示。

内容	描述
创建资源栈	创建资源栈时，您需要设置模板、配置模板参数，并配置资源栈策略、删除保护等内容。
查看资源栈	您可以在资源栈管理页面查看资源栈基本信息。
更新资源栈	当您需要修改当前的资源栈模板、资源栈配置，不需要修改资源栈的所属地域时，您可以更新资源栈。
克隆资源栈	当您需要修改当前的资源栈模板、资源栈配置、资源栈的所属地域时，您可以重建资源栈。
删除资源栈	当您不需要资源栈以及其资源时，可以从指定地域内的指定目标账号中删除资源栈及其所有关联资源。
启用删除保护	为了防止资源栈被意外删除，您可以启用删除保护。此时针对资源栈的删除操作将会失败，资源栈状态保持不变。
替换更新资源	如果您需要更新资源，但是资源的属性不支持直接修改，您可以替换更新资源。
使用嵌套资源栈	嵌套资源栈本身可以包含其他嵌套资源栈，构成一个资源栈层次结构。根资源栈是所有嵌套资源栈最终归属的父资源栈。您可以通过资源编排控制台查看嵌套资源栈的父资源栈。
资源栈策略	创建资源栈时，允许对所有资源执行更新操作，具有资源栈更新权限的用户可以更新资源栈中的所有资源。更新时，一些资源可能需要中断。使用资源栈策略可以防止资源栈资源在资源栈更新过程中被意外更新或删除。

3.2. 创建资源栈

资源编排服务ROS（Resource Orchestration Service）可以通过创建资源栈来管理一组资源。本文为您介绍如何创建资源栈。

背景信息

创建资源栈有以下两种方式：

- 标准：创建新资源。
  当您使用已有模板或示例模板创建资源栈时，默认创建新资源。具体操作，请参见使用新资源创建资源栈。
- 资源导入：使用现有资源。
  您可以将现有资源通过资源导入的方式，导入到ROS的资源栈中，统一进行资源管理和编排。具体操作，请参见使用现有资源创建资源栈。

使用新资源创建资源栈

1. 登录资源编排控制台。
2. 在左侧导航栏，单击**资源栈**。
3. 在页面左上角的地域下拉列表，选择资源栈的所在地域。
4. 在**资源栈列表**页面，单击**创建资源栈**，然后在下拉列表中选择**使用新资源（标准）**。
5. 在**选择模板**页面，根据需要指定模板，然后单击**下一步**。
   - **选择已有模板**：选择模板录入方式，然后设置模板。
     - **使用url**：输入模板对应的URL，然后单击**获取JSON内容**，系统自动填入模板内容。
     - **输入模板**：在**模板内容**区域输入JSON、YAML或Terraform三种类型的模板。关于如何编写模板，JSON或YAML格式请参见模板结构说明，Terraform类型请参见Terraform类型模板结构。
     - **我的模板**：选择已有模板和模板版本，系统自动填入模板内容。关于如何创建模板，请参见创建模板和使用可视化编辑器编写模板。
     - **共享模板**：选择他人共享给您的模板和模板版本，系统自动填入模板内容。关于如何添加共享，请参见将模板共享给阿里云账号。
     - **上传模板**：单击**上传文件**，上传JSON或YAML类型的模板，系统自动填入模板内容。
   - **使用示例模板**：选择示例模板，系统自动填入模板内容。
6. 在**配置模板参数**页面，配置**资源栈名称**及参数，然后单击**下一步**。

   > **说明** 根据模板的不同，您需要输入的参数将有所差异，请根据控制台提示输入参数信息。

7. 在**配置资源栈**页面，配置以下参数，然后单击**下一步**。

配置项	说明
资源栈策略	取值： ○ **无资源栈策略**：不设置资源栈策略。 ○ **输入资源栈策略**：上传文件或在文本框手动输入资源栈策略。 关于资源栈策略的更多信息，请参见资源栈策略。
失败时回滚	取值： ○ **已启用**：创建资源栈失败时，启用回滚策略。 ○ **已禁用**：创建资源栈失败时，禁用回滚策略。
超时设置	如果所有资源的创建或更新没有在该时间内完成，系统将自动回滚到创建或更新之前的状态。 取值范围：10~1440。 单位：分钟。
删除保护	防止资源栈被意外删除。取值： ○ **已启用**：启用删除保护。 ○ **已禁用**：禁用删除保护。

配置项	说明
RAM角色	您可以创建可信实体为资源编排服务的RAM角色，然后根据ROS模板中资源所需最小权限为RAM角色授权。 ○ 如果指定RAM角色，ROS将根据RAM角色的权限创建资源栈。关于如何创建RAM角色和为RAM角色授权，请参见创建可信实体为阿里云服务的RAM角色和为RAM角色授权。 ○ 如果不指定RAM角色，ROS将使用当前账号相关权限创建资源栈。
标签	由一对键值对组成，方便您对资源栈进行分类。 您可以单击**添加**，然后在**编辑标签绑定**对话框设置**标签键**和**标签值**，最后单击**确定**。
资源组	您可以选择资源栈所在的资源组。如果不指定资源组，资源栈将加入默认资源组。 关于如何创建资源组，请参见创建资源组。

8. 在**检查并确认**页面，单击**创建**。

   资源栈创建成功后，**状态**列显示**创建成功**。

使用现有资源创建资源栈

下文以导入弹性公网IP（EIP）资源为例，为您介绍使用现有资源创建资源栈的操作方法。

前提条件

- 获取EIP资源的标识符属性。
  本示例中，获取到的EIP资源的标识符属性为AllocationId，即EIP的实例ID。具体操作，请参见获取待导入资源的标识符属性。
- 获取EIP的实例ID。
  登录EIP控制台，获取要导入的EIP的实例ID。

操作步骤

1. 登录资源编排控制台。
2. 在左侧导航栏，单击**资源栈**。
3. 在页面左上角的地域下拉列表，选择资源栈的所在地域。

   > **说明** 请确保待导入资源与资源栈处于同一地域。

4. 在**资源栈列表**页面，单击**创建资源栈**，然后在下拉列表中选择**使用现有资源（资源导入）**。
5. 在**选择模板**页面的**指定模板**区域，选择**选择已有模板**，设置**模板录入方式**为**输入模板**，并在**模板内容**区域输入如下JSON格式的模板，然后单击**下一步**。

```
{
  "ROSTemplateFormatVersion": "2015-09-01",
  "Resources": {
    "Eip": {
      "Type": "ALIYUN::VPC::EIP",
      "DeletionPolicy": "Retain",
      "Properties": {
        "Bandwidth": 5
      }
    }
  },
  "Outputs": {
    "EipAddress": {
      "Value": {
        "Fn::GetAtt": [
          "Eip",
          "EipAddress"
        ]
      }
    },
    "AllocationId": {
      "Value": {
        "Fn::GetAtt": [
          "Eip",
          "AllocationId"
        ]
      }
    }
  }
}
```

说明 `DeletionPolicy`：取值为`Retain`，表示删除保护策略为保留资源。为防止资源被误删除，请务必设置该项。

6. 在**识别资源**页面，输入资源标识符值（例如：`eip-bp140qv3j25nsfaqd****`），单击**下一步**。

7. 在**配置模板参数**页面，配置**资源栈名称**和**更改集名称**，单击**下一步**。
8. 在**配置资源栈**页面，配置相关参数，单击**下一步**。

   本示例使用默认配置。更多信息，请参见参数说明。

9. 在**检查并确认**页面，单击**创建资源栈和导入更改集**。
10. 在**更改集**页签，单击更改集右侧**操作**列的**执行**，执行更改集，开始资源导入。

执行结果

在**资源栈信息**页签，如果**基本信息**区域的**状态**显示为**导入创建完成**，则资源导入成功。

后续步骤

您可以在**资源栈信息**页签，单击**偏差状态**右侧的**检测偏差**，检测导入资源的模板与实际模板的匹配情况。具体操作，请参见检测资源栈的偏差状态。

3.3. 继续创建资源栈

如果您创建资源栈时禁用了失败时回滚，当资源栈创建失败时可以继续创建资源栈。

操作步骤

1. 登录资源编排控制台。
2. 在左侧导航栏，单击**资源栈**。
3. 在页面左上角的地域下拉列表，选择资源栈的所在地域。
4. 在**资源栈列表**页面，单击目标资源栈**操作**列的**继续创建**。
5. 在**继续创建**对话框，根据失败原因和实际需求选择继续创建模式，然后单击**确定**。
   - **继续创建**：当失败原因为外部因素（例如：超过配额、服务不可用、流量控制、库存不足、欠费等）时，可以选择**继续创建**，ROS将继续创建资源栈。
   - **丢弃**（可选）：当您不需要创建失败和未创建的资源时，可以选择**丢弃**，忽略并丢弃所有创建失败和未创建的资源，直接把资源栈状态标记为成功。

     > **说明** 仅ROS类型的资源栈支持选择丢弃，Terraform类型的资源栈不支持选择丢弃。

   - **高级选项**：当模板内容或参数设置不正确时，可以选择**高级选项**，根据需求更新模板或参数值。
     a. （可选）在**进行设置**页面，选择需要清空并再次创建的资源，然后单击**下一步**。

        > **说明** 仅ROS类型的资源栈支持设置资源，Terraform类型的资源栈不支持设置资源。

     b. 在**模板设置**页面，选择**使用当前模板**或**替换当前模板**，然后单击**下一步**。
        关于替换当前模板时如何选择模板，请参见选择模板。
     c. 在**参数设置**页面，更新参数值，然后单击**下一步**。

d. 在**配置资源栈**页面，设置**资源最大并发数**（可选）和**RAM角色**，然后单击**下一步**。

> **说明** 仅Terraform类型的资源栈支持设置资源最大并发数，ROS类型的资源栈不支持设置资源最大并发数。

e. 在**检查并确认**页面，确认继续创建时设置的模板、参数、RAM角色等信息，然后单击**继续创建**。

3.4. 查看资源栈

本文为您介绍如何查看资源栈信息，包括资源栈基本信息、策略、事件和资源等。

前提条件

- 请确保您已创建资源栈，具体操作请参见创建资源栈。
- 如果您需要查看该资源栈下的更改集，请确保您已经创建了更改集。具体操作请参见创建更改集。
- 如果您需要查看该资源栈的偏差状态，请确保您已经完成了资源栈的偏差检测。具体操作请参见检测资源栈的偏差状态。
- 仅当模板中指定了输出信息时，您才可以查看资源栈的输出信息。

操作步骤

1. 登录资源编排控制台。
2. 在左侧导航栏单击**资源栈**。
3. 在页面左上角的地域下拉列表，选择资源栈的所在地域。
4. 在**资源栈列表**页面，找到需要查看的资源栈，单击**资源栈名称**列的资源栈ID。

   在资源栈管理页面，您可以执行以下操作：
   - 单击**资源栈信息**页签，查看**基本信息**和**资源栈策略**。
   - 单击**事件**页签，查看资源栈生命周期中发生的每一个事件。
   - 单击**资源**页签，查看资源栈所包括的每一个资源的信息。
   - 单击**输出**页签，查看创建资源栈时，模板中指定的输出信息。
   - 单击**参数**页签，查看创建资源栈时，模板中指定的参数，包括ROS提供的以`ALIYUN::`开始的伪参数。
   - 单击**偏差**页签，查看**资源栈偏差状态**和**资源偏差状态**。
   - 单击**模板**页签，查看资源栈所对应的模板信息。
   - 单击**更改集**页签，查看该资源栈下的更改集。

3.5. 更新资源栈

当您需要更改当前资源栈的模板或模板参数时，可以使用更新资源栈的操作。本文为您介绍如何通过资源编排服务ROS（Resource Orchestration Service）更新资源栈。

前提条件

请确保您已创建资源栈，具体操作请参见创建资源栈。

使用限制

只有以下状态的资源栈支持更新资源栈：

状态	说明
CREATE_COMPLETE	资源栈创建成功。
UPDATE_FAILED	资源栈更新失败。
UPDATE_COMPLETE	资源栈更新完成。
ROLLBACK_COMPLETE	资源栈回滚完成。
ROLLBACK_FAILED	资源栈回滚失败。
IMPORT_CREATE_COMPLETE	通过资源导入创建资源栈成功。
IMPORT_UPDATE_COMPLETE	通过资源导入更新资源栈成功。
IMPORT_UPDATE_FAILED	通过资源导入更新资源栈失败。
IMPORT_UPDATE_ROLLBACK_COMPLETE	通过资源导入更新资源栈失败，回滚成功。
IMPORT_UPDATE_ROLLBACK_FAILED	通过资源导入更新资源栈失败，回滚失败。
CHECK_FAILED	资源栈校验失败。
CHECK_COMPLETE	资源栈校验完成。

操作步骤

1. 登录资源编排控制台。
2. 在左侧导航栏，单击**资源栈**。
3. 在页面左上角的地域下拉列表，选择资源栈的所在地域。
4. 在**资源栈列表**页面，单击目标资源栈右侧**操作**列的**更新**。
5. 在**配置模板参数**页面，根据控制台提示，重新配置参数。

   > **说明** 如果您需要修改资源栈的模板，单击**上一步**，进入**选择模板**页面，然后在**准备模板**区域选择**替换当前模板**。模板修改完成后，再单击**下一步**，进入**配置模板参数**页面，根据控制台提示，配置参数。

6. （可选）单击**下一步**，在**配置资源栈**页面，根据控制台提示，配置**资源栈策略**、**超时设置**、**RAM角色**、**是否启用替换更新**和**标签**。

   > **说明**
   > - 如果您需要配置资源栈策略，可选择**输入资源栈策略**，那么下次更新创建时，即使您不做设置，本次的策略仍然生效；或者您也可以选择**输入临时资源栈策略**，那么资源栈策略仅在本次更新创建时生效。
   > - 资源栈策略是一个JSON或YAML类型的文档，具体请参见资源栈策略。

7. （可选）单击**下一步**，检查并确认更新配置。
8. 单击**确认修改**。

3.6. 克隆资源栈

当您需要基于现有资源栈的模板和参数重新创建资源栈时，可以使用克隆资源栈。本文为您介绍如何通过资源编排服务ROS（Resource Orchestration Service）克隆资源栈。

前提条件

确保您已创建资源栈，具体操作请参见创建资源栈。

使用限制

只有以下状态的资源栈支持克隆资源栈：

状态	说明
CREATE_COMPLETE	资源栈创建成功。
UPDATE_COMPLETE	资源栈更新完成。
ROLLBACK_COMPLETE	资源栈回滚完成。
IMPORT_CREATE_COMPLETE	通过资源导入创建资源栈成功。
IMPORT_UPDATE_COMPLETE	通过资源导入更新资源栈成功。
IMPORT_UPDATE_ROLLBACK_COMPLETE	通过资源导入更新资源栈失败，回滚成功。
CHECK_COMPLETE	资源栈校验完成。

操作步骤

1. 登录资源编排控制台。
2. 在左侧导航栏，单击**资源栈**。
3. 在页面左上角的地域下拉列表，选择资源栈的所在地域。
4. 在**资源栈列表**页面，在目标资源栈右侧**操作**列选择 > **克隆**。
5. 在**配置模板参数**页面，配置**资源栈名称**和模板参数。
6. （可选）单击**下一步**，在**配置资源栈**页面，配置以下参数。

配置项	说明
资源栈策略	取值： ○ **无资源栈策略**：不设置资源栈策略。 ○ **输入资源栈策略**：上传文件或在文本框手动输入资源栈策略。 关于资源栈策略的更多信息，请参见资源栈策略。
失败时回滚	取值： ○ **已启用**：创建资源栈失败时，启用回滚策略。 ○ **已禁用**：创建资源栈失败时，禁用回滚策略。

配置项	说明
超时设置	如果所有资源的创建或更新没有在该时间内完成，系统将自动回滚到创建或更新之前的状态。 取值范围：10~1440。 单位：分钟。
删除保护	防止资源栈被意外删除。取值： ○ **已启用**：启用删除保护。 ○ **已禁用**：禁用删除保护。
RAM角色	您可以创建可信实体为资源编排服务的RAM角色，然后根据ROS模板中资源所需最小权限为RAM角色授权。 ○ 如果指定RAM角色，ROS将根据RAM角色的权限创建资源栈。关于如何创建RAM角色和为RAM角色授权，请参见创建可信实体为阿里云服务的RAM角色和为RAM角色授权。 ○ 如果不指定RAM角色，ROS将使用当前账号相关权限创建资源栈。
标签	由一对键值对组成，方便您对资源栈进行分类。 您可以单击**添加**，然后在**编辑标签绑定**对话框设置**标签键**和**标签值**，最后单击**确定**。
资源组	您可以选择资源栈所在的资源组。如果不指定资源组，资源栈将加入默认资源组。 关于如何创建资源组，请参见创建资源组。

7. （可选）单击**下一步**，检查并确认资源栈配置。
8. 单击**创建**。

3.7. 删除资源栈

本文为您介绍如何删除资源栈。

前提条件

- 请确保资源栈没有处于正在创建中、更新中或者其它操作中。
- 如果您使用RAM用户删除资源栈，请确保该RAM用户具有AliyunROSFullAccess权限。具体操作，请参见为RAM用户授权。

操作步骤

1. 登录资源编排控制台。
2. 在左侧导航栏单击**资源栈**。
3. 在页面左上角的地域下拉列表，选择资源栈的所在地域。
4. 在**资源栈列表**页面，找到需要删除的资源栈，单击右侧**操作**列中的**删除**。
5. 在**删除资源栈**对话框，选择**RAM角色**。
   - 如果指定RAM角色，ROS将使用RAM角色的权限删除资源栈。
   - 如果不指定RAM角色，ROS将使用当前账号的权限删除资源栈。
6. 选择**删除方式**。

- **保留资源**：删除资源栈时保留资源栈中的资源。您可以从**资源项**下拉列表中选择需要保留的资源。
- **释放资源**：删除资源栈时释放资源栈中的资源，请您谨慎操作。

7. 单击**确定**。

相关文档

- 删除资源栈失败怎么办？

3.8. 启用删除保护

删除保护是资源编排服务ROS（Resource Orchestration Service）针对资源栈的一种保护措施，防止资源栈被意外删除。当您启用删除保护时，针对资源栈的删除操作将会失败，资源栈状态保持不变。

前提条件

请确保您已经具有设置删除保护的权限（ros:SetDeletionProtection）。更多信息，请参见使用RAM控制资源访问。

背景信息

您不能对处于删除状态的资源栈启用删除保护。嵌套资源栈的删除保护属性由其父资源栈决定，始终与父资源栈一致，且无法被修改。如果您需要对嵌套资源栈启用删除保护，则需要在其父资源栈进行操作。

在创建资源栈时启用删除保护

1. 登录资源编排控制台。
2. 在左侧导航栏，单击**资源栈**。
3. 在页面左上角的地域下拉列表，选择资源栈的所在地域。
4. 在**资源栈列表**页面，单击**创建资源栈**，然后在下拉列表中选择**使用新资源（标准）**。
5. 在**选择模板**页面，根据需要选择模板，然后单击**下一步**。
6. 在**配置模板参数**页面，根据控制台提示，配置**资源栈名称**及模板参数，然后单击**下一步**。
7. 在**配置资源栈**页面，设置**删除保护**。
   - **已启用**：启用删除保护。
   - **已禁用**：禁用删除保护。
8. 单击**下一步**。
9. 在**检查并确认**页面，单击**创建**。

在已有资源栈启用删除保护

1. 登录资源编排控制台。
2. 在左侧导航栏，单击**资源栈**。
3. 在页面左上角的地域下拉列表，选择资源栈的所在地域。
4. 在**资源栈列表**页面，单击目标资源栈**资源栈名称**列的资源栈ID。
5. 在**资源栈信息**页签的**基本信息**区域，打开**删除保护**开关，启用删除保护。

> **说明** 您也可以关闭**删除保护**开关，禁用删除保护。

在嵌套资源栈上启用删除保护

嵌套资源栈不能直接启用删除保护，您只能在其父资源栈进行操作。

1. 登录资源编排控制台。
2. 在左侧导航栏，单击**资源栈**。
3. 在页面左上角的地域下拉列表，选择资源栈的所在地域。
4. 在**资源栈列表**页面，勾选**显示嵌套资源栈**。
5. 在**资源栈列表**页面，单击目标资源栈**资源栈名称**列的资源栈ID。
6. 在**资源栈信息**页签的**基本信息**区域，单击**父资源栈ID**。
7. 在父资源栈**资源栈信息**页签的**基本信息**区域，打开**删除保护**开关，启用删除保护。

> **说明** 您也可以关闭**删除保护**开关，禁用删除保护。

3.9. 替换更新资源

如果您需要更新资源，但是资源的属性不支持直接修改，您可以替换更新资源。

背景信息

如果您只需要修改资源的属性，保留原有的资源（资源物理ID不变），您可以在资源栈模板中修改参数属性。

如果您需要更新资源，但是资源的属性不支持直接修改，您可以通过替换更新功能删除资源后重新创建资源，此时资源物理ID会发生变化。本文以替换 `ALIYUN::ECS::VSwitch` 的属性CidrBlock为例，为您介绍如何替换更新资源。

操作步骤

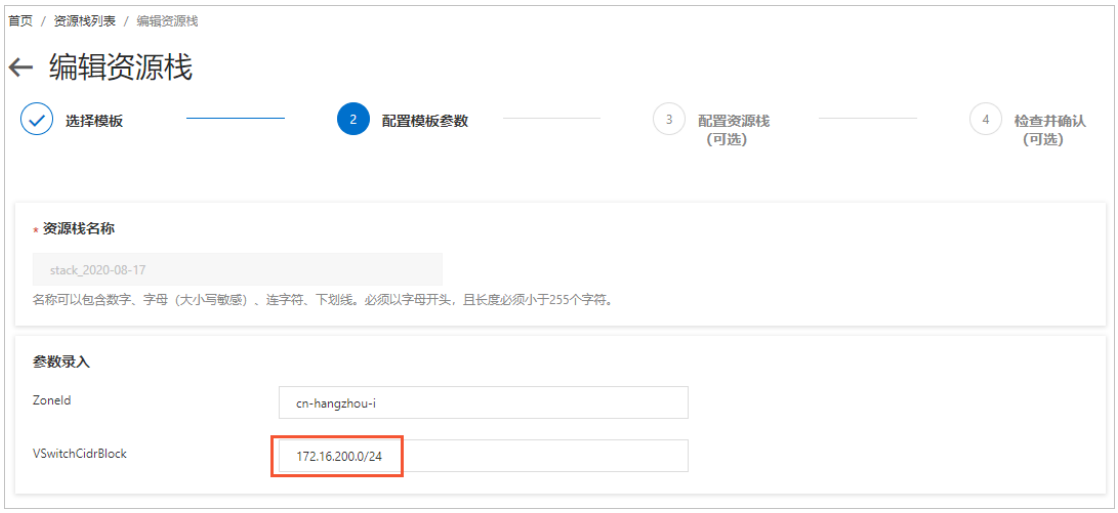
1. 登录资源编排控制台。
2. 创建资源栈。

   使用如下模板创建一个包含 `ALIYUN::ECS::VSwitch` 资源的资源栈，属性CidrBlock取值为172.16.100.0/24。
   关于如何创建资源栈，请参见创建资源栈。

```
{
    "ROSTemplateFormatVersion": "2015-09-01",
    "Parameters": {
        "ZoneId": {
            "Type": "String",
            "Default": "cn-hangzhou-i"
        },
        "VSwitchCidrBlock": {
            "Type": "String",
            "Default": "172.16.100.0/24"
        }
    },
    "Resources": {
        "EcsVpc": {
            "Type": "ALIYUN::ECS::VPC",
            "Properties": {
                "CidrBlock": "172.16.0.0/12",
                "VpcName": "MyTestVpc"
            }
        },
        "VSwitch": {
            "Type": "ALIYUN::ECS::VSwitch",
            "Properties": {
                "ZoneId": {
                    "Ref": "ZoneId"
                },
                "CidrBlock": {
                    "Ref": "VSwitchCidrBlock"
                },
                "VpcId": {
                    "Fn::GetAtt": [
                        "EcsVpc",
                        "VpcId"
                    ]
                },
                "VSwitchName": "VSwitch"
            }
        }
    },
    "Outputs": {
    }
}
```

3. 替换更新资源栈。
   i. 在左侧导航栏，单击**资源栈**。
   ii. 在页面左上角的地域下拉列表，选择资源栈的所在地域。
   iii. 在**资源栈列表**页面，找到需要更新的资源栈，单击右侧**操作**列中的**更新**。

iv. 在**配置模板参数**页面的**参数录入**区域，将 `VSwitchCidrBlock` 由172.16.100.0/24更新为172.16.200.0/24。



v. 单击**下一步**。

vi. 在**配置资源栈**页面，单击**启用**，开启替换更新功能。

vii. 单击**确认修改**。

替换更新成功后，vSwitch资源物理ID将发生变化，CidrBlock参数将从172.16.100.0/24替换为172.16.200.0/24。您可以在资源栈详情页单击**资源**页签，单击新的vSwitch资源ID进入交换机详情页查看资源信息。

3.10. 使用嵌套资源栈

本文为您介绍嵌套资源栈的结构、最佳实践、常见模板、更新行为和输出值，以及如何查看嵌套资源栈及其所属的父资源栈。

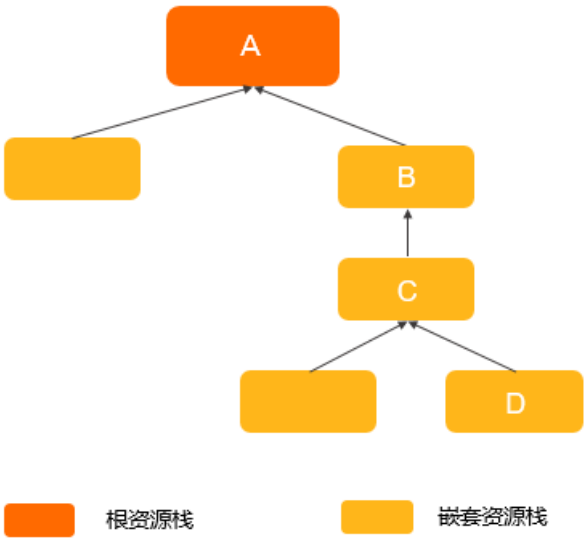
嵌套资源栈的结构

嵌套资源栈本身可以包含其他嵌套资源栈，构成一个资源栈层次结构。根资源栈是所有嵌套资源栈最终归属的父资源栈。

每个嵌套资源栈都有一个直属父资源栈。对于第一级的嵌套资源栈而言，根资源栈也是父资源栈。

嵌套资源栈和根资源栈的关系如下：

- 资源栈A是该层次结构中所有其他嵌套资源栈的根资源栈。
- 对于资源栈B来说，资源栈A既是父资源栈，也是根资源栈。
- 对于资源栈D，资源栈C是父资源栈；而对于资源栈C来说，资源栈B是父资源栈。



某些资源栈操作（如资源栈更新等）应从根资源栈启动，而不是直接在嵌套资源栈上执行。此外，在某些情况下，嵌套资源栈会影响资源栈操作的执行。

最佳实践

资源编排之嵌套资源栈

使用嵌套资源栈来重复使用常见模板

随着基础设施的发展，常见模板模式可合并以便声明每个模板中的相同组件。您可以分离这些常见组件并为其创建专用模板，混合和匹配不同的模板，但使用嵌套资源栈来创建单个统一资源栈。

嵌套资源栈是作为其他资源栈的一部分来创建的资源栈。您可以在另一个资源栈中使用ALIYUN::ROS::Stack资源创建嵌套资源栈。ALIYUN::ROS::Stack资源详情，请参见ALIYUN::ROS::Stack。

例如：您有用于大多数资源栈的负载均衡器配置。您可以为负载均衡器创建专用模板，而不是将相同的配置复制并粘贴到您的模板中。然后，您只需使用ALIYUN::ROS::Stack资源从其他模板中引用该模板。当您更新该资源栈后，如果更新负载均衡器模板，引用该模板的资源栈将使用更新过的负载均衡器。

嵌套资源栈资源的更新行为

如果模板包括多个嵌套资源栈，则ROS会为每个嵌套资源栈启动更新，以便您确定嵌套资源栈是否已修改。

ROS只更新嵌套资源栈中在相应模板指定了更改的资源。

使用嵌套资源栈的输出值

嵌套资源栈是您使用ALIYUN::ROS::Stack资源在其他资源栈中创建的资源栈。使用嵌套资源栈，您可从一个资源栈部署和管理所有资源。您可以将来自嵌套资源栈组中的一个资源栈的输出用作该组中的另一个资源栈的输入。

查看属于父资源栈的嵌套资源栈

1. 登录资源编排控制台。
2. 在左侧导航栏单击**资源栈**。
3. 在**资源栈列表**页面，找到需要查看其嵌套资源栈的父资源栈，单击**资源栈名称**列的资源栈ID。

   > **说明** 如果父资源栈也是嵌套资源栈，需要勾选**显示嵌套资源栈**。

4. 单击**资源**页签。
   查找类型为ALIYUN::ROS::Stack的资源。

查看嵌套资源栈的父资源栈

1. 登录资源编排控制台。
2. 在左侧导航栏单击**资源栈**。
3. 在**资源栈列表**页面，勾选**显示嵌套资源栈**，查看资源栈列表。
4. 找到需要查看父资源栈的嵌套资源栈，单击**资源栈名称**列的资源栈ID。
5. 单击**资源栈信息**页签，查看父资源栈ID。

3.11. 资源栈策略

使用资源栈策略可以防止资源栈资源在资源栈更新过程中被意外更新或删除。本文为您介绍资源栈策略的定义，以及如何设置和更新资源栈策略。

背景信息

资源栈策略是一个JSON或YAML类型的文档，该文档定义可对指定资源执行的更新操作。创建资源栈时，允许对所有资源执行更新操作，具有资源栈更新权限的用户可以更新资源栈中的所有资源。更新时，一些资源可能需要中断。设置资源栈策略后，ROS将保护资源栈中的所有资源。您可以在资源栈策略中为资源指定明确的 `Allow` 语句，允许对特定资源进行更新。

> **说明**
> - 您只能为每个资源栈定义一个资源栈策略，但一个策略可以保护多个资源。
> - 在资源栈更新期间，ROS自动更新依赖其他更新的资源。例如：ROS自动更新引用更新的资源。但如果这些资源与资源栈策略关联，您必须具有权限才能进行更新。

资源栈策略仅在资源栈更新过程中适用。与RAM策略不同，它不提供访问控制，仅将资源栈策略用作故障保护功能来防止意外更新特定资源栈资源。

定义资源栈策略

创建资源栈时，如果未设置资源栈策略，则允许对所有资源执行更新操作。要阻止对资源栈资源执行更新操作，可定义一个资源栈策略，然后对资源栈设置该策略。在创建资源栈时，您可以指定一个包含资源栈策略的文本文件或输入该策略来设置资源栈策略。在资源栈上设置资源栈策略时，默认情况下会拒绝未显式允许的任何更新。

您可定义一个带5个元素的资源栈策略：`Effect`、`Action`、`Principal`、`Resource`和`Condition`。

```
{
  "Statement" : [
    {
      "Effect" : "Deny_or_Allow",
      "Action" : "update_actions",
      "Principal" : "*",
      "Resource" : "LogicalResourceId/resource_logical_ID",
      "Condition" : {
        "StringEquals_or_StringLike" : {
          "ResourceType" : [resource_type, ...]
        }
      }
    }
  ]
}
```

各元素说明如下：

- `Effect`
  确定是拒绝还是允许对指定资源执行指定的操作。您只能指定`Deny`或`Allow`，例如：

  ```
  "Effect" : "Deny"
  ```

  > **说明** 如果资源栈策略包含重叠语句（同时允许和拒绝对资源进行更新），则`Deny`语句始终将覆盖`Allow`语句。要确保某一资源受到保护，请对该资源使用`Deny`语句。

- `Action`
  拒绝或允许的更新操作：
  - `Update:Modify`
    在对资源应用更改期间不会中断或有某些中断的更新操作。
  - `Update:Delete`
    删除资源的更新操作。从资源栈模板中完全删除资源的更新都需要此操作。
  - `Update:*`
    所有更新操作。星号（*）是通配符，代表所有更新操作。

  > **说明** `Action`还可以指定`Update:Replace`作为保留功能。目前暂不支持替换功能。

  以下示例说明如何只指定更新和删除操作：

  ```
  "Action" : ["Update:Modify", "Update:Delete"]
  ```

  要允许除某个更新操作之外的所有更新操作，请使用`NotAction`。例如：要允许除`Update:Delete`之外的所有更新操作，请使用`NotAction`。

```
{
  "Statement" : [
    {
      "Effect" : "Allow",
      "NotAction" : "Update:Delete",
      "Principal": "*",
      "Resource" : "*"
    }
  ]
}
```

- `Principal`
  策略应用的实体。仅支持星号（*），表示策略应用于所有主体。
- `Resource`
  应用策略的资源的逻辑ID。要指定资源类型，请使用 `Condition` 元素。
  要指定一个资源，请使用其逻辑ID。例如：

  ```
  "Resource" : ["LogicalResourceId/myECS"]
  ```

  您可以对逻辑ID使用星号（*）。例如：如果您对所有相关资源使用一个通用逻辑ID前缀，则可使用星号（*）指定所有资源。

  ```
  "Resource" : ["LogicalResourceId/Prefix*"]
  ```

  您还可以对资源使用 `Not` 元素。例如：要允许对所有资源执行除某个更新之外的所有更新，请使用 `NotResource` 元素保护该资源。

  ```
  {
    "Statement" : [
      {
        "Effect" : "Allow",
        "Action" : "Update:*",
        "Principal": "*",
        "NotResource" : "LogicalResourceId/WebServers"
      }
    ]
  }
  ```

  设置资源栈策略时，会拒绝未显式允许的任何更新。通过允许更新 `WebServers` 资源之外的所有资源，会拒绝更新 `WebServers` 资源。
- `Condition`
  应用策略的资源类型。要指定特定资源的逻辑ID，请使用 `Resource` 元素。
  您可以指定资源类型（例如：所有ECS和RDS数据库实例）。

```
{
  "Statement" : [
  {
    "Effect" : "Deny",
    "Principal" : "*",
    "Action" : "Update:*",
    "Resource" : "*",
    "Condition" : {
      "StringEquals" : {
        "ResourceType" : ["ALIYUN::ECS::Instance", "ALIYUN::RDS::DBInstance"]
      }
    }
  },
  {
    "Effect" : "Allow",
    "Principal" : "*",
    "Action" : "Update:*",
    "Resource" : "*"
  }
  ]
}
```

`Allow` 语句授予对所有资源的更新权限，而 `Deny` 语句拒绝对ECS和RDS数据库实例的更新。 `Deny` 语句始终覆盖允许操作。

您可以对资源类型使用星号（*）。例如：您可以使用星号（*）拒绝所有ECS资源（例如：实例、安全组和子网）的更新权限。

```
"Condition" : {
  "StringLike" : {
    "ResourceType" : ["ALIYUN::ECS::*"]
  }
}
```

> **说明** 使用星号（*）时，必须使用StringLike条件。

设置资源栈策略

您可以使用控制台或ALIYUN CLI在创建资源栈时应用资源栈策略。您也可以使用ALIYUN CLI将资源栈策略应用于现有资源栈。应用资源栈策略后，无法将其从资源栈中删除，但可以使用ALIUN CLI进行更新。

- 在创建资源栈时设置资源栈策略（控制台）
  i. 登录资源编排控制台。
  ii. 在左侧导航栏单击**资源栈**。
  iii. 在页面左上角的地域下拉列表，选择资源栈的所在地域。
  iv. 在**资源栈列表**页面，单击**创建资源栈**，然后在下拉列表中选择**使用新资源（标准）**。
  v. 在**创建资源栈**向导的**选择模板**页面，根据所需选择模板，然后单击**下一步**。
  vi. 在**创建资源栈**向导的**配置模板参数**页面，根据控制台提示，配置**资源栈名称**和**参数录入**，然后单击**下一步**。
  vii. 在**创建资源栈**向导的**配置资源栈**页面，选择**资源栈策略**为**输入资源栈策略**。

viii. 配置资源栈策略。
- **输入资源栈策略**：直接输入资源栈策略。
- **上传文件**：上传包含资源栈策略的JSON或YAML格式的文件。

ix. 按照控制台提示继续配置，完成资源栈创建。

- 在创建资源栈时设置资源栈策略（CLI）
  您可以使用以下两种方式设置资源栈策略：
  - 调用CreateStack接口设置
    使用 `aliyun ros CreateStack` 和 `--StackPolicyBody` 设置可更新的策略，或使用 `aliyun ros CreateStack` 和 `--StackPolicyURL` 指定包含策略的文件。
  - 调用CreateChangeSet接口设置
    使用 `aliyun ros CreateChangeSet` 和 `--StackPolicyBody` 设置可更新的策略，或使用 `aliyun ros CreateChangeSet` 和 `--StackPolicyURL` 指定包含策略的文件。
- 在现有资源栈上设置资源栈策略（仅限CLI）
  使用 `aliyun ros SetStackPolicy` 和 `--StackPolicyBody` 设置可更新的策略，或使用 `aliyun ros SetStackPolicy` 和 `--StackPolicyURL` 指定包含策略的文件。

> **说明** 要将策略添加到现有资源栈中，您必须具有 `SetStackPolicy` 操作权限。

更新受保护资源

当您需要更新受保护资源时，可以创建一个覆盖资源栈策略并允许对这些资源进行更新的临时策略。覆盖策略不会永久更改资源栈策略。

要更新保护的资源，您必须具有 `SetStackPolicy` 操作权限。关于如何设置ROS权限，请参见使用RAM控制资源访问。

- 更新受保护的资源（控制台）
  i. 登录资源编排控制台。
  ii. 在左侧导航栏单击**资源栈**。
  iii. 在页面左上角的地域下拉列表，选择资源栈的所在地域。
  iv. 在**资源栈列表**页面，单击目标资源栈右侧**操作**列的**更新**。
  v. 在**配置模板参数**页面，单击**下一步**。
  vi. 在**配置资源栈**页面，根据控制台提示，选择**输入临时资源栈策略**。

vii. 配置临时资源栈策略。
指定临时的资源栈策略，仅本次更新生效。覆盖策略必须为您要更新的受保护资源指定 `Allow` 语句。例如：要更新所有受保护资源，可以指定允许所有更新的临时覆盖策略。

```
{
  "Statement" : [
    {
      "Effect" : "Allow",
      "Action" : "Update:*",
      "Principal": "*",
      "Resource" : "*"
    }
  ]
}
```

viii. 按照控制台提示继续配置，完成资源栈更新。

- 更新受保护资源（CLI）
  您可以通过以下两种方式更新受保护资源：
  - 调用UpdateStack接口更新
    使用 `aliyun ros UpdateStack` 和 `--StackPolicyDuringUpdateBody` 设置可更新的策略，或使用 `aliyun ros UpdateStack` 和 `--StackPolicyDuringUpdateURL` 指定包含策略的文件。
  - 调用CreateChangeSet接口更新
    使用 `aliyun ros CreateChangeSet` 和 `--StackPolicyDuringUpdateBody` 设置可更新的策略，或使用 `aliyun ros CreateChangeSet` 和 `--StackPolicyDuringUpdateURL` 指定包含策略的文件。

> **说明** ROS仅在此更新期间应用覆盖策略。覆盖策略不会永久更改资源栈策略。

更新资源栈策略

当您需要保护其他资源或从资源中删除保护时，可以更新资源栈策略。例如：当您将要保护的数据库添加到资源栈时，会将该数据库的 `Deny` 语句添加到资源栈策略。如果需要更新策略，您必须具有 `SetStackPolicy` 的使用权限。

- 更新资源栈策略（控制台）

i. 登录资源编排控制台。
ii. 在左侧导航栏单击**资源栈**。
iii. 在页面左上角的地域下拉列表，选择资源栈的所在地域。
iv. 在**资源栈列表**页面，单击目标资源栈**资源栈名称**列的资源栈ID。
v. 在**资源栈信息**页签的**资源栈策略**区域，单击**编辑**。
vi. 在**修改资源栈策略**对话框，输入资源栈策略。
vii. 单击**确定**。

- 更新资源栈策略（CLI）
  使用 `aliyun ros SetStackPolicy` 和 `--StackPolicyBody` 设置更新的策略，或使用 `aliyun ros SetStackPolicy` 和 `--StackPolicyURL` 指定包含策略的文件。
  以下策略允许对所有资源进行全部更新：

```
{
  "Statement" : [
    {
      "Effect" : "Allow",
      "Action" : "Update:*",
      "Principal": "*",
      "Resource" : "*"
    }
  ]
}
```

- 在更新资源栈时更新资源栈策略（CLI）
  使用 `aliyun ros UpdateStack` 和 `--StackPolicyBody` 设置可更新的策略，或使用 `aliyun ros UpdateStack` 和 `--StackPolicyURL` 指定包含策略的文件。
  使用 `aliyun ros CreateChangeSet` 和 `--StackPolicyBody` 设置可更新的策略，或使用 `aliyun ros CreateChangeSet` 和 `--StackPolicyURL` 指定包含策略的文件。

资源栈策略示例

以下示例策略说明如何阻止对所有资源栈资源和特定资源进行更新，并阻止特定类型的更新。

- 阻止对所有资源栈资源的更新
  要阻止对所有资源栈资源的更新，以下策略为所有资源的所有更新操作指定 `Deny` 语句。

```
{
  "Statement" : [
    {
      "Effect" : "Deny",
      "Action" : "Update:*",
      "Principal": "*",
      "Resource" : "*"
    }
  ]
}
```

- 阻止对单个资源（ `WebServers` ）的更新

- 示例一：使用 `Deny` 语句阻止对 `WebServers` 资源的更新。

```
{
  "Statement" : [
    {
      "Effect" : "Allow",
      "Action" : "Update:*",
      "Principal": "*",
      "Resource" : "*"
    },
    {
      "Effect" : "Deny",
      "Action" : "Update:*",
      "Principal": "*",
      "Resource" : "LogicalResourceId/WebServers"
    }
  ]
}
```

  各元素说明如下：
  - `Allow`：允许对所有资源执行的操作。
  - `Deny`：为具有 `WebServers` 逻辑ID的资源阻止执行的操作。
  - `Principal`：策略应用的实体。仅支持星号（*），表示策略应用于所有实体。

- 示例二：使用 `Allow` 语句允许对 `WebServers` 之外的所有资源进行更新。

```
{
  "Statement" : [
    {
      "Effect" : "Allow",
      "Action" : "Update:*",
      "Principal": "*",
      "NotResource" : "LogicalResourceId/WebServers"
    }
  ]
}
```

  说明
  - 设置资源栈策略时，如果对某个资源没有写明是否支持更新，则默认不支持更新该资源。
  - 使用默认拒绝存在风险。如果您策略中的其他位置具有 `Allow` 语句（例如：使用通配符的 `Allow` 语句），则可能意外授予对资源的更新权限。由于显示拒绝将覆盖任何允许操作，因此可以使用 `Deny` 语句确保保护资源。

- 阻止对资源类型的所有实例进行更新
  以下策略拒绝针对RDS数据库实例资源类型的所有更新操作。使用 `Allow` 语句允许对所有其他资源栈资源进行全部更新操作。 `Allow` 语句不应用于RDS数据库实例资源，因为 `Deny` 语句始终覆盖允许操作。

```
{
  "Statement" : [
    {
      "Effect" : "Deny",
      "Action" : "Update:*",
      "Principal": "*",
      "Resource" : "*",
      "Condition" : {
        "StringEquals" : {
          "ResourceType" : ["ALIYUN::RDS::DBInstance"]
        }
      }
    },
    {
      "Effect" : "Allow",
      "Action" : "Update:*",
      "Principal": "*",
      "Resource" : "*"
    }
  ]
}
```

- 阻止对嵌套资源栈进行更新
  以下策略拒绝针对ROS资源栈资源类型（嵌套资源栈）的所有更新操作。使用 `Allow` 语句允许对所有其他资源栈资源进行全部更新操作。 `Allow` 语句不会应用于ROS资源栈资源，因为 `Deny` 语句始终覆盖 `Allow` 操作。

```
{
  "Statement" : [
    {
      "Effect" : "Deny",
      "Action" : "Update:*",
      "Principal": "*",
      "Resource" : "*",
      "Condition" : {
        "StringEquals" : {
          "ResourceType" : ["ALIYUN::ROS::Stack"]
        }
      }
    },
    {
      "Effect" : "Allow",
      "Action" : "Update:*",
      "Principal": "*",
      "Resource" : "*"
    }
  ]
}
```

3.12. 事件通知

资源编排服务ROS（Resource Orchestration Service）支持事件通知，用于传递资源变化信息。

背景信息

背景信息

云监控的事件监控功能提供了各阿里云产品系统事件的统一查询和统计入口，使您明确得知其使用状态。当阿里云产品发生系统异常时，事件监控为您提供事件报警功能，方便您及时知晓事件的发生，并自动化处理异常。ROS已接入云监控，当资源栈执行状态发生变化时，会对云监控发出事件通知，方便云监控查询事件和报警。

系统事件使用详情，请参见：

- 查看系统事件
- 创建系统事件报警规则

通知状态

ROS支持的资源栈通知状态码如下：

- CREATE_FAILED
- CREATE_COMPLETE
- UPDATE_FAILED
- UPDATE_COMPLETE
- DELETE_FAILED
- DELETE_COMPLETE
- ROLLBACK_FAILED
- ROLLBACK_COMPLETE
- CREATE_ROLLBACK_FAILED
- CREATE_ROLLBACK_COMPLETE

通知格式

以CREATE_COMPLETE状态为例，ROS事件通知格式如下：

```
{
    "ver": "1.0",
    "id": "939F2DC4-BA3E-4539-BB95-03A300D16****",
    "product": "ROS",
    "resourceId": "acs:ros:cn-hangzhou:151266687691****:stack/37811dc0-9c3a-4112-ba1e-b1988
106****",
    "level": "INFO",
    "name": "Stack:StatusChange",
    "userId": "151266687691****",
    "eventTime": "20200616T220114.058+0800",
    "regionId": "cn-hangzhou",
    "content": {
        "callerId": "29154787464691****",
        "requestId": "939F2DC4-BA3E-4539-BB95-03A300D52467",
        "stackId": "37811dc0-9c3a-4112-ba1e-b1988106****",
        "stackName": "stack_2020-06-18stack",
        "status": "CREATE_COMPLETE",
        "statusReason": "Stack CREATE completed successfully"
    }
}
```

事件属性

事件通知包含的字段及其含义如下表所示。

字段	描述	示例
id	事件ID。	939F2DC4-BA3E-4539-BB95-03A300D16****
eventTime	事件发生时间，采用UTC+8时区。	20200616T220114.058+0800
level	事件级别。	INFO
name	事件名称。	Stack:StatusChange
product	产品名称。取值：ROS。	ROS
regionId	阿里云地域ID。	cn-hangzhou
resourceId	资源的ARN值（Aliyun Resource Name，阿里云全局资源描述符）。	acs:ros:cn-hangzhou:151266687691****:stack/37811dc0-9c3a-4112-ba1e-b1988106****
userId	阿里云账号ID。	151266687691****
content	事件详情。包含一个或多个参数。	无

content包含的字段及其含义如下表所示。

字段	描述	示例
stackId	资源栈ID。	37811dc0-9c3a-4112-ba1e-b1988106****
stackName	资源栈名称。	stack_2020-06-18stack
status	资源栈状态。	CREATE_COMPLETE
statusReason	资源栈状态的原因。	Stack CREATE completed successfully
callerId	执行此次资源栈操作的账号ID。（如果是阿里云账号操作，则为阿里云账号ID；如果是RAM用户操作，则为RAM用户ID）。	29154787464691****
requestId	请求ID。	939F2DC4-BA3E-4539-BB95-03A300D52467

4.资源栈组

4.1. 概览

资源栈组可以帮助您高效、低成本管理多个资源栈。您可以使用同一模板跨账号、跨地域创建多个资源栈，实现账号维度或资源夹维度对资源栈的统一部署。

基本概念

概念	说明
管理员账号	用于创建资源栈组的账号。管理员账号可以是阿里云账号、资源目录中的企业管理账号或委派管理员账号。具体如下： • 当您使用阿里云账号创建自助管理权限模式的资源栈组时，ROS将在其他阿里云账号中部署资源栈。该场景中管理员账号和目标账号都是阿里云账号。 • 当您开通了资源目录，使用资源目录的企业管理账号或委派管理员账号创建服务管理权限模式的资源栈组时，ROS将在目标账号中部署资源栈。该场景中管理员账号是资源目录的企业管理账号或委派管理员账号，目标账号是资源目录的成员账号。
目标账号	资源栈部署生效的账号。

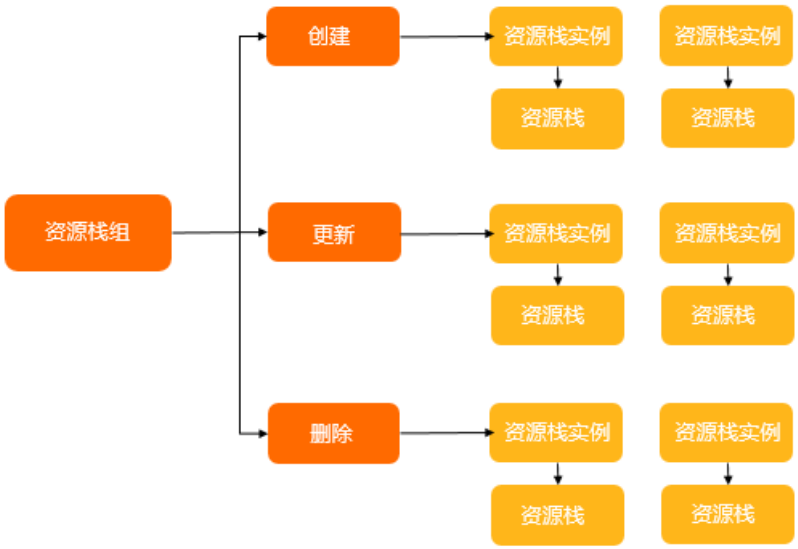
授权模式

当您创建自助管理权限模式和服务管理权限模式的资源栈组时，需要使用指定权限。授权模式如下表所示：

授权模式	说明	操作步骤
自助管理权限	当您创建自助管理权限模式的资源栈组时，需要事先在管理员账号和目标账号中手动创建RAM角色，建立二者的信任关系，然后在其他阿里云账号中部署资源栈。	• 步骤一：授权自助管理权限 • 步骤二：创建资源栈组 • 步骤三（可选）：创建资源栈实例
服务管理权限	当您创建服务管理权限模式的资源栈组时，只需开启可信访问，ROS会为管理员账号和目标账号自动创建服务关联角色，管理员账号通过服务关联角色在目标账号中部署资源栈。	• 步骤一（可选）：设置委派管理员账号 • 步骤二：开启可信访问 • 步骤三：创建资源栈组 • 步骤四（可选）：创建资源栈实例

工作原理

当您在管理员账号中创建资源栈组时，会在指定的目标账号和地域下创建资源栈实例，从而创建与资源栈实例对应的资源栈。在资源栈组中的更新、删除操作也将影响对应的资源栈实例和资源栈。资源栈组、资源栈实例和资源栈之间的关系如下：

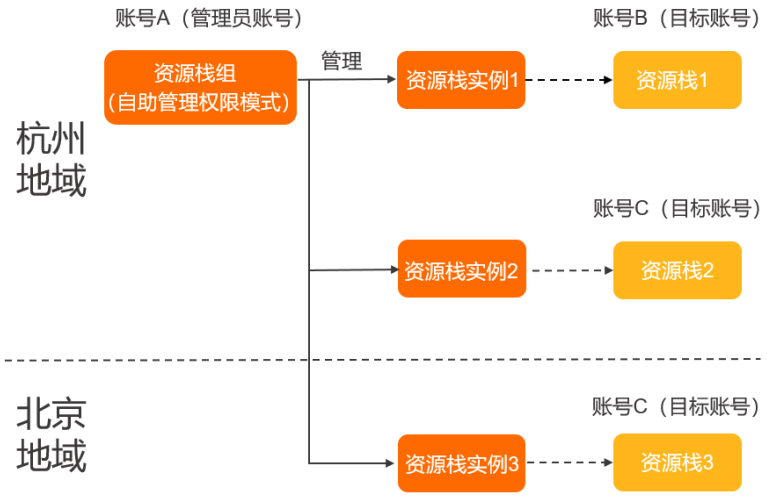


资源栈组、资源栈实例和资源栈的使用说明如下：

- 一个资源栈实例只属于一个资源栈组。
- 一个资源栈实例对应一个或零个资源栈。如果您因为某些原因无法创建资源栈，可能出现资源栈实例存在，而对应的资源栈不存在的情况，此时资源栈实例将显示资源栈创建失败的原因。
- 删除资源栈实例时可选择删除或保留它所指向的资源栈。
- 直接删除资源栈不会删除资源栈实例。

管理员账号可以创建自助管理权限模式和服务管理权限模式的资源栈组，在目标账号中跨账号、跨地域创建资源栈。

- 自助管理权限模式的资源栈组：管理员账号（账号A）在杭州地域创建了一个自助管理权限模式的资源栈组，在杭州地域为目标账号（账号B和账号C）、北京地域为目标账号（账号C）创建资源栈。



- 服务管理权限模式的资源栈组：管理员账号（账号A）在杭州地域创建了一个服务管理权限模式的资源栈组，在杭州地域和北京地域分别为资源目录中指定的资源夹创建资源栈。ROS会自动获取该资源夹中的所

有成员账号（账号B和账号C）作为目标账号。

资源栈组、地域、权限、账号的使用说明如下：

- 资源栈组仅在管理员账号所在地域产生。例如：您在杭州地域创建了资源栈组，不会在北京地域同步创建。
- 当您创建自助管理权限模式和服务管理权限模式的资源栈组时，需要建立管理员账号和目标账号间的信任关系，然后跨账号创建资源栈。关于权限的更多信息，请参见授权模式。
- 当您创建服务管理权限模式的资源栈组时，可以开启自动部署。当资源目录中有新账号加入或者账号移出时，ROS将会自动进行资源栈实例的创建或删除。

资源栈部署设置

在创建资源栈组或资源栈实例时，您可以配置下表所示的资源栈组参数，从而控制资源栈部署时间和失败次数，快速创建多个资源栈。

参数	说明	示例
最大并发账号数（MaxConcurrentCount）	每个地域中可同时部署资源栈的账号数。	如果您需要将资源栈部署到2个地域内的5个目标账号，可以将最大并发账号数设置为3，此时ROS会在第一个地域为前3个账号部署资源栈，完成后在第一个地域为另外2个账号部署资源栈，随后转入下一个地域执行同样的操作。
最大并发账号百分比（MaxConcurrentPercentage）	每个地域中可同时部署资源栈的账号数占总账号数的百分比。 说明 百分比所得账号数不为整数时ROS会向下取整。	如果您需要将资源栈部署到5个目标账号，可以将最大并发账号百分比设置为50，此时ROS会同时部署2个资源栈。
容错数（FailureToleranceCount）	每个地域中资源栈可以失败的账号数，超过该数字将停止该地域中的操作。	如果您需要将资源栈部署到2个地域内的5个目标账号，可以将失败容错数设置为2，如果某个地域中有第3个目标账号创建资源栈失败，则ROS将停止本次操作，本次操作的结果为失败。如果两个地域内创建资源栈失败的目标账号数都小于等于2，则本次操作结果为成功。

参数	说明	示例
容错百分比（FailureTolerancePercentage）	每个地域中资源栈可以失败的账号数占总账号数的百分比，超过该百分比将停止该地域中的操作。 说明 百分比所得账号数不为整数时ROS会向下取整。	如果您需要将资源栈部署到5个目标账号，可以将失败容错百分比设置为50，则ROS最多允许2个目标账号创建资源栈失败。
地域并发类型（RegionConcurrencyType）	部署资源栈实例的地域级别并发类型。 • 顺序（默认值）：根据地域顺序依次在指定的每个地域部署，同一时间只在一个地域部署。 • 并行：在所有指定区域内并行部署。	如果您需要将资源栈部署到2个地域内的5个目标账号，并将最大并发账号数设置为3，将地域并发类型设置为并行，此时ROS会在2个地域内同时为前3个账号部署资源栈，完成后在2个地域内为剩余的2个账号部署资源栈。

4.2. 使用自助管理权限模式部署资源栈

4.2.1. 步骤一：授权自助管理权限

当您创建自助管理权限模式的资源栈组时，需要事先在管理员账号和目标账号中手动创建RAM角色，建立二者的信任关系，然后在其他阿里云账号中部署资源栈。

背景信息

授权自助管理权限时，您需要为下表所示的两个阿里云账号分别创建RAM角色并授权：

阿里云账号	RAM角色	权限策略	权限策略说明
管理员账号（账号A）	AliyunROSStackGroupAdministrationRole	自定义策略AssumeRole-AliyunROSStackGroupExecutionRole	允许RAM角色（AliyunROSStackGroupAdministrationRole）扮演角色身份（AliyunROSStackGroupExecutionRole）。
目标账号（账号B）	AliyunROSStackGroupExecutionRole	系统策略AdministratorAccess	允许RAM角色（AliyunROSStackGroupExecutionRole）管理目标账号（账号B）的所有阿里云资源。

说明 管理员账号和目标账号可以为同一阿里云账号。关于管理员账号和目标账号的更多信息，请参见基本概念。

授权成功后，当您使用管理员账号（账号A）登录资源编排控制台创建资源栈组后，即可在该资源栈组中为目标账号（账号B）创建资源栈。

方式一：通过资源编排控制台设置权限

1. 设置目标账号（账号B）的权限。
   i. 使用目标账号（账号B）登录RAM控制台。
   ii. 为目标账号（账号B）创建可信实体为管理员账号（账号A）的RAM角色（AliyunROSStackGroupExecutionRole）。
      a. 在左侧导航栏，选择**身份管理 > 角色**。
      b. 在**角色**页面，单击**创建角色**。
      c. 在**创建角色**面板，选择可信实体类型为**阿里云账号**，然后单击**下一步**。
      d. 输入**角色名称**为**AliyunROSStackGroupExecutionRole**，然后选择**其他云账号**，最后输入管理员账号（账号A）的ID。
      e. 单击**完成**。
   iii. 为RAM角色（AliyunROSStackGroupExecutionRole）授予**AdministratorAccess**权限。
      a. 在**创建角色**面板，单击**为角色授权**。
      b. 在**添加权限**面板，授权主体会自动填入，选择**授权范围**为**整个云账号**。
      c. 选择权限为**系统策略**，然后选择**AdministratorAccess**。
      d. 单击**确定**。
      e. 单击**完成**。
2. 设置管理员账号（账号A）的权限。
   i. 使用管理员账号（账号A）登录RAM控制台。
   ii. 为管理员账号（账号A）创建可信实体为资源编排服务的RAM角色（AliyunROSStackGroupAdministrationRole）。
      a. 在左侧导航栏，选择**身份管理 > 角色**。
      b. 在**角色**页面，单击**创建角色**。
      c. 在**创建角色**面板，选择可信实体类型为**阿里云服务**，然后单击**下一步**。
      d. 选择**角色类型**为**普通服务角色**。
      e. 输入**角色名称**为**AliyunROSStackGroupAdministrationRole**，然后选择受信服务为**资源编排服务**。
      f. 单击**完成**。
      g. 单击**关闭**。

iii. 创建自定义权限策略（AssumeRole-AliyunROSStackGroupExecutionRole）。

a. 在左侧导航栏，选择**权限管理 > 权限策略管理**。

b. 在**权限策略管理**页面，单击**创建权限策略**。

c. 在**新建自定义权限策略**页面，输入**策略名称**为**AssumeRole-AliyunROSStackGroupExecutionRole**，然后选择**配置模式**为**脚本配置**，最后输入以下策略内容。
该策略允许RAM角色（AliyunROSStackGroupAdministrationRole）扮演角色身份（AliyunROSStackGroupExecutionRole）。

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "sts:AssumeRole",
      "Resource": "acs:ram::*:role/AliyunROSStackGroupExecutionRole"
    }
  ],
  "Version": "1"
}
```

d. 单击**确定**。

iv. 为RAM角色（AliyunROSStackGroupAdministrationRole）授予**AssumeRole-AliyunROSStackGroupExecutionRole**权限。

a. 在左侧导航栏，选择**身份管理 > 角色**。

b. 在**角色**页面，单击RAM角色（AliyunROSStackGroupAdministrationRole）**操作**列的**添加权限**。

c. 在**添加权限**面板，授权主体会自动填入，选择**授权范围**为**整个云账号**。

d. 选择权限为**自定义策略**，然后选择**AssumeRole-AliyunROSStackGroupExecutionRole**。

e. 单击**确定**。

f. 单击**完成**。

方式二：通过资源编排模板设置权限

通过资源编排模板为管理员账号（账号A）和目标账号（账号B）创建RAM角色，并赋予资源栈组和资源栈的操作权限。

1. 使用管理员账号（账号A）登录资源编排控制台。
2. 使用模板AliyunROSStackGroupAdministrationRole为管理员账号（账号A）创建RAM角色并授权。
3. 使用模板AliyunROSStackGroupExecutionRole为目标账号（账号B）创建RAM角色并授权。

4.2.2. 步骤二：创建资源栈组

当您需要跨账号、跨地域部署多个资源栈时，可以创建资源栈组，在指定的账号和地域下创建资源栈实例，从而创建与资源栈实例对应的资源栈。

背景信息

创建资源栈组时，您需要准备以下账号：

- 管理员账号（账号A）：用于创建资源栈组的账号。
- 目标账号（账号B）：资源栈部署生效的账号。

> **说明** 管理员账号和目标账号可以为同一阿里云账号。关于管理员账号和目标账号的更多信息，请参见基本概念。

本文以管理员账号（账号A）创建资源栈组，为目标账号（账号B）在杭州和北京地域部署资源栈为例，为您进行介绍。

操作步骤

1. 使用管理员账号（账号A）登录资源编排控制台。
2. 在左侧导航栏，单击**资源栈组**。
3. 在页面左上角的地域下拉列表，选择要创建资源栈组的地域。
4. 在**资源栈组列表**页面，单击**创建资源栈组**。
5. 在**选择模板**页面，选择已有模板或者使用示例模板，然后单击**下一步**。
6. 在**配置模板参数**页面，输入**资源栈组名称**和**资源栈组描述**，然后配置参数，最后单击**下一步**。

   > **说明** 根据模板的不同，您需要输入的参数将有所差异，请根据控制台提示输入参数信息。

7. 在**配置资源栈组**页面，选择**自助管理权限**，然后设置**管理员角色**（AliyunROSStackGroupAdministrationRole）和**执行角色**（AliyunROSStackGroupExecutionRole），最后单击**下一步**。

   > **说明** 管理员角色和执行角色为授权自助管理权限时创建的RAM角色。更多信息，请参见步骤一：授权自助管理权限。

8. 在**设置部署选项**页面，设置以下参数，然后单击**下一步**。

参数	说明
账号	资源栈部署生效的账号。 请根据需求选择部署位置，然后设置相应的账号。 ○ **在账号中部署资源栈**：当您使用阿里云账号创建资源栈组时，可以选择在账号中部署资源栈，然后在其他阿里云账号中部署资源栈。例如：输入目标账号（账号B），在目标账号（账号B）中部署资源栈。 > **说明** 您可以添加多个账号，多个账号之间用半角逗号（,）分隔。您也可以添加管理员账号（账号A），在管理员账号（账号A）中部署资源栈。 ○ **在资源目录中部署资源栈**：当您开通了资源目录，使用资源目录的企业管理账号创建资源栈组时，可以选择在资源目录中部署资源栈，然后选择资源目录的资源夹，在资源夹的所有成员账号中部署资源栈。
地域	要部署资源栈的地域。例如：华东1（杭州）和华北2（北京）。
资源组	用于分组管理资源栈组的组织。 请根据需求选择资源栈组所在的资源组。如果不指定资源组，资源栈组将加入默认资源组。 关于如何创建资源组，请参见创建资源组。

参数	说明
最大并发账号数	每个地域中可同时部署资源栈的账号数。 关于如何设置最大并发账号数，请参见资源栈部署设置。
容错	每个地域中资源栈可以失败的账号数，超过该数字将停止该地域中的操作。如果停止了一个地域中的操作，此操作在其他地域中不继续执行。 关于如何设置容错，请参见资源栈部署设置。
区域并发	要部署资源栈实例的区域级别并发类型。 ○ 顺序：根据区域顺序依次在指定的每个区域部署，同一时间只在一个区域部署。 ○ 并发：在所有指定区域内并行部署。

> **说明** 如果不指定**账号**和**地域**，则ROS只创建资源栈组，不创建源栈实例。如果需要在指定账号和地域部署资源栈，您还需创建资源栈实例。具体操作，请参见步骤三（可选）：创建资源栈实例。

9. 在**检查并确认**页面，检查资源栈组信息无误后，单击**创建资源栈组**。

执行结果

资源栈组创建成功后，您可以在**资源栈组列表**页面查看管理员账号（账号A）中的资源栈组。

单击资源栈组名称，然后单击**实例**页签，可以查看目标账号（账号B）的资源栈实例状态。当实例状态为**最新**时，资源栈部署成功。此时您可以使用目标账号（账号B）登录资源编排控制台，查看杭州地域和北京地域已部署的资源栈。

4.2.3. 步骤三（可选）：创建资源栈实例

您可以在已有资源栈组中创建资源栈实例，指定资源栈实例所在账号和地域，以便跨账号、跨地域部署资源栈。如果创建资源栈组时指定了账号和地域，将自动创建资源栈实例，此时您也可以再次创建资源栈实例，部署更多资源栈。

背景信息

创建资源栈实例时，您需要准备以下账号：

- 管理员账号（账号A）：用于创建资源栈实例的账号，该账号与资源栈实例所在资源栈组的创建账号相同。
- 目标账号（账号B）：资源栈部署生效的账号。

> **说明** 管理员账号和目标账号可以为同一阿里云账号。关于管理员账号和目标账号的更多信息，请参见基本概念。

本文以管理员账号（账号A）创建资源栈实例，为目标账号（账号B）在杭州和北京地域部署资源栈为例，为您进行介绍。

操作步骤

1. 使用管理员账号（账号A）登录资源编排控制台。
2. 在左侧导航栏，单击**资源栈组**。
3. 在页面左上角的地域下拉列表，选择资源栈组所在的地域。

4. 在**资源栈组列表**页面，在目标资源栈组**操作**列选择 > **创建资源栈实例**。

5. 在**设置部署选项**页面，设置以下参数，然后单击**下一步**。

参数	说明
账号	资源栈部署生效的账号。 请根据需求选择部署位置，然后设置相应的账号。 ○ **在账号中部署资源栈**：当您使用阿里云账号创建资源栈实例时，可以选择在账号中部署资源栈，然后在其他阿里云账号中部署资源栈。例如：输入目标账号（账号B），在目标账号（账号B）中部署资源栈。 **说明** 您可以添加多个账号，多个账号之间用半角逗号（,）分隔。您也可以添加管理员账号（账号A），在管理员账号（账号A）中部署资源栈。 ○ **在资源目录中部署资源栈**：当您开通了资源目录，使用资源目录的企业管理账号创建资源栈实例时，可以选择在资源目录中部署资源栈，然后选择资源目录的资源夹，在资源夹的所有成员账号中部署资源栈。
地域	要部署资源栈的地域。例如：华东1（杭州）和华北2（北京）。
最大并发账号数	每个地域中可同时部署资源栈的账号数。 关于如何设置最大并发账号数，请参见资源栈部署设置。
容错	每个地域中资源栈可以失败的账号数，超过该数字将停止该地域中的操作。如果停止了一个地域中的操作，此操作在其他地域中不继续执行。 关于如何设置容错，请参见资源栈部署设置。
区域并发	要部署资源栈实例的区域级别并发类型。 ○ 顺序：根据区域顺序依次在指定的每个区域部署，同一时间只在一个区域部署。 ○ 并发：在所有指定区域内并行部署。

6. 在**指定覆盖**页面，选择目标参数，使用以下两种方式修改参数值，然后单击**下一步**。
   - 方式一：选择**编辑覆盖值** > **覆盖资源栈组参数值**对参数进行重置。
   - 方式二：选择**编辑覆盖值** > **重置覆盖字段**输入新的参数值。

   说明 资源栈实例创建成功后，您可以在目标账号（账号B）的**资源栈列表**，单击资源栈ID，然后在**参数**页签下，查看重新指定的参数值。

7. 在**检查并确认**页面，确认资源栈实例无误后，单击**确认添加**。

执行结果

资源栈实例创建后，您可以在**资源栈组列表**页面单击目标资源栈组名称，然后单击**实例**页签，查看目标账号（账号B）的资源栈实例状态。

资源栈部署过程中，实例状态为**已过期**。资源栈部署成功后，实例状态自动更新为**最新**，此时您可以使用目标账号（账号B）登录资源编排控制台，查看杭州地域和北京地域已部署的资源栈。

4.3. 使用服务管理权限模式部署资源栈

4.3.1. 步骤一（可选）：设置委派管理员账号

通过委派管理员账号，可以将组织管理任务与业务管理任务相分离，企业管理账号执行资源目录的组织管理任务，委派管理员账号执行ROS的业务管理任务，这符合安全最佳实践的建议。您可以根据需要，将资源目录的成员账号设置为委派管理员账号，从而作为ROS的管理员账号为成员账号部署资源栈。

前提条件

- 请确保您已经开通资源目录。具体操作，请参见开通资源目录。
- 请确保您已经在资源目录中创建成员或邀请成员。具体操作，请参见创建成员和邀请阿里云账号加入资源目录。

背景信息

为ROS设置委派管理员账号后，委派管理员账号将获得企业管理账号的授权，获取在ROS中访问资源目录中组织和成员信息，以及管理ROS资源栈组、资源栈实例、资源栈等功能的权限。
更多信息，请参见企业管理账号和委派管理员账号。

添加委派管理员账号

您可以使用企业管理账号登录资源管理控制台，在资源目录中为ROS添加委派管理员账号。具体操作，请参见添加委派管理员账号。

> **说明** ROS最多支持添加5个委派管理员账号。

移除委派管理员账号

委派管理员账号不建议随意移除，否则将导致一些配置暂时失效。如果需要移除委派管理员账号，请参见移除委派管理员账号。

移除委派管理员账号后，原有委派管理员账号作为普通的成员账号，将发生以下变化：

- 原有委派管理员账号中的资源栈组和资源栈实例不会被自动删除。
- 原有委派管理员账号不能对具有服务管理权限的资源栈组中的资源栈实例进行创建、更新、删除等操作。
- 原有委派管理员账号可信访问权限将自动关闭。

4.3.2. 步骤二：开启可信访问

当您开通了资源目录，使用资源目录的企业管理账号或委派管理员账号创建服务管理权限模式的资源栈组时，必须开启可信访问，授权服务管理权限。

操作步骤

> **说明** 仅在首次创建服务管理权限模式的资源栈组时，需要开启可信访问。后续创建无需再次开启。

1. 使用企业管理账号或委派管理员账号登录资源编排控制台。
2. 在左侧导航栏，单击**资源栈组**。
3. 在**资源栈组列表**页面，单击右上角的**开启可信访问**。

4.3.3. 步骤三：创建资源栈组

当您开通了资源目录，使用资源目录的企业管理账号或委派管理员账号创建服务管理权限模式的资源栈组时，ROS将使用服务管理权限在成员账号中部署资源栈。

操作步骤

1. 使用资源目录的企业管理账号或委派管理员账号登录资源编排控制台。
2. 在左侧导航栏，单击**资源栈组**。
3. 在页面左上角的地域下拉列表，选择要创建资源栈组的地域。
4. 在**资源栈组列表**页面，单击**创建资源栈组**。
5. 在**选择模板**页面，选择已有模板或者使用示例模板，然后单击**下一步**。
6. 在**配置模板参数**页面，输入**资源栈组名称**和**资源栈组描述**，然后配置参数，最后单击**下一步**。

   > **说明** 根据模板的不同，您需要输入的参数将有所差异，请根据控制台提示输入参数信息。

7. 在**配置资源栈组**页面，选择**服务管理权限**，然后单击**下一步**。
8. 在**设置部署选项**页面，配置以下参数，然后单击**下一步**。

参数	说明
资源目录	您可以选择资源目录的资源夹，在资源夹的所有成员账号中部署资源栈。 如果您选择的是Root资源夹，则会在整个资源目录的所有成员账号中部署资源栈。
自动部署	目标资源夹中成员账号变动时，是否自动部署资源栈实例。取值： ○ 已启用：启用自动部署后，如果目标资源夹中新增了成员账号，资源栈组会自动将资源栈实例部署到该账号。如果目标资源夹中删除了成员账号，则资源栈组会自动删除该账号中的资源栈实例。 ○ 已禁用：禁用自动部署后，目标资源夹中成员账号变动时资源栈实例不会发生变化。 **说明** 您也可以在创建资源栈组**详细信息**页签编辑自动部署，修改自动部署选项。具体操作，请参见自动部署资源栈实例。
账号删除行为	目标资源夹中删除成员账号时，是否删除成员账号中的资源栈。取值： ○ 删除资源栈：目标资源夹中删除成员账号时，同步删除成员账号中的资源栈。 ○ 保留资源栈：目标资源夹中删除成员账号时，保留成员账号中的资源栈。
地域	要部署资源栈的地域。
资源组	用于分组管理资源栈组的组织。 请根据需求选择资源栈组所在的资源组。如果不指定资源组，资源栈组将加入默认资源组。 关于如何创建资源组，请参见创建资源组。
最大并发账号数	每个地域中可同时部署资源栈的账号数。 关于如何设置最大并发账号数，请参见资源栈部署设置。
容错	每个地域中资源栈可以失败的账号数，超过该数字将停止该地域中的操作。如果停止了一个地域中的操作，此操作在其他地域中不继续执行。 关于如何设置容错，请参见资源栈部署设置。

参数	说明
区域并发	要部署资源栈实例的区域级别并发类型。 ○ 顺序：根据区域顺序依次在指定的每个区域部署，同一时间只在一个区域部署。 ○ 并发：在所有指定区域内并行部署。

> **说明** 如果不指定**账号**和**地域**，则ROS只创建资源栈组，不创建源栈实例。如果需要在指定账号和地域部署资源栈，您还需创建资源栈实例。具体操作，请参见步骤四（可选）：创建资源栈实例。

9. 在**检查并确认**页面，检查资源栈组信息无误后，单击**创建资源栈组**。

执行结果

资源栈组创建成功后，您可以在**资源栈组列表**页面查看资源目录的企业管理账号或委派管理员账号中的资源栈组。

单击资源栈组名称，然后单击**实例**页签，可以查看成员账号的资源栈实例状态。当实例状态为**最新**时，资源栈部署成功。此时您可以使用成员账号登录资源编排控制台，查看各地域已部署的资源栈。成员账号登录方法，请参见访问成员。

4.3.4. 步骤四（可选）：创建资源栈实例

您可以在已有资源栈组中创建资源栈实例，指定资源栈实例所在账号和地域，以便跨账号、跨地域部署资源栈。如果创建资源栈组时指定了账号和地域，将自动创建资源栈实例，此时您也可以再次创建资源栈实例，部署更多资源栈。

操作步骤

1. 使用资源目录的企业管理账号或委派管理员账号登录资源编排控制台。

   > **说明** 登录账号需要与创建资源栈组的账号相同。

2. 在左侧导航栏，单击**资源栈组**。
3. 在页面左上角的地域下拉列表，选择资源栈组所在的地域。
4. 在**资源栈组列表**页面，在目标资源栈组**操作**列选择 > **创建资源栈实例**。

5. 在**设置部署选项**页面，设置以下参数，然后单击**下一步**。

参数	说明
资源目录	您可以选择资源目录的资源夹，在资源夹的所有成员账号中部署资源栈。 如果您添加了Root资源夹，将在整个资源目录的所有成员账号中创建资源栈实例。
地域	要部署资源栈的地域。
最大并发账号数	每个地域中可同时部署资源栈的账号数。 关于如何设置最大并发账号数，请参见资源栈部署设置。

参数	说明
容错	每个地域中资源栈可以失败的账号数，超过该数字将停止该地域中的操作。如果停止了一个地域中的操作，此操作在其他地域中不继续执行。 关于如何设置容错，请参见资源栈部署设置。
区域并发	要部署资源栈实例的区域级别并发类型。 ○ 顺序：根据区域顺序依次在指定的每个区域部署，同一时间只在一个区域部署。 ○ 并发：在所有指定区域内并行部署。

6. 在**指定覆盖**页面，选择目标参数，使用以下两种方式修改参数值，然后单击**下一步**。
   - 方式一：选择**编辑覆盖值 > 覆盖资源栈组参数值**对参数进行重置。
   - 方式二：选择**编辑覆盖值 > 重置覆盖字段**输入新的参数值。

   > **说明** 资源栈实例创建成功后，您可以在目标账号（账号B）的**资源栈列表**，单击资源栈ID，然后在**参数**页签下，查看重新指定的参数值。

7. 在**检查并确认**页面，确认资源栈实例无误后，单击**确认添加**。

执行结果

资源栈实例创建后，您可以在**资源栈组列表**页面单击目标资源栈组名称，然后单击**实例**页签，查看成员账号的资源栈实例状态。

资源栈部署过程中，实例状态为**已过期**。资源栈部署成功后，实例状态自动更新为**最新**，此时您可以使用成员账号登录资源编排控制台，查看各地域已部署的资源栈。成员账号登录方法，请参见访问成员。

4.3.5. 资源编排服务关联角色

本文为您介绍资源编排服务ROS（Resource Orchestration Service）的服务关联角色（AliyunServiceRoleForROSStackGroupsRDAdmin和AliyunServiceRoleForROSStackGroupsRDMember）的应用场景、权限策略及相关操作。

应用场景

ROS服务关联角色（AliyunServiceRoleForROSStackGroupsRDAdmin和AliyunServiceRoleForROSStackGroupsRDMember）是在使用具有服务管理权限模式的资源栈组情况下，为了获取管理员账号下的资源目录账号和对成员账号部署资源栈，需要获取其他云服务的访问权限，而提供的RAM角色。

关于服务关联角色的更多信息，请参见服务关联角色。

权限说明

- **AliyunServiceRoleForROSStackGroupsRDAdmin**
  权限策略：AliyunServiceRolePolicyForROSStackGroupsRDAdmin。
  权限说明：ROS使用此角色来获取资源目录的账号信息。

```
{
  "Statement": [
    {
      "Action": [
        "resourcemanager:ListAccountsForParent",
        "resourcemanager:ListFoldersForParent",
        "resourcemanager:ListAncestors"
      ],
      "Effect": "Allow",
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": "sts:AssumeRole",
      "Resource": "acs:ram:*:*:role/stackgroups-exec-*"
    },
    {
      "Action": "ram:DeleteServiceLinkedRole",
      "Resource": "*",
      "Effect": "Allow",
      "Condition": {
        "StringEquals": {
          "ram:ServiceName": "stackgroups-admin.ros.aliyuncs.com"
        }
      }
    }
  ],
  "Version": "1
```

- **AliyunServiceRoleForROSStackGroupsRDMember**
  权限策略：AliyunServiceRolePolicyForROSStackGroupsRDMember。
  权限内容：ROS使用此角色来动态创建 `stackgroups-exec-` 开头的RAM角色，并以此身份部署资源栈。

```
{
  "Statement": [
    {
      "Action": [
          "ram:CreateRole",
          "ram:GetRole",
          "ram:DeleteRole"
        ],
      "Effect": "Allow",
      "Resource": "acs:ram:*:*:role/stackgroups-exec-*"
    },
    {
      "Action": [
          "ram:AttachPolicyToRole",
          "ram:DetachPolicyFromRole"
        ],
      "Effect": "Allow",
      "Resource": [
        "acs:ram:*:*:role/stackgroups-exec-*",
        "acs:ram:*:system:policy/AdministratorAccess"
      ]
    },
    {
      "Action": "ram:DeleteServiceLinkedRole",
      "Resource": "*",
      "Effect": "Allow",
      "Condition": {
        "StringEquals": {
          "ram:ServiceName": "stackgroups-member.ros.aliyuncs.com"
        }
      }
    }
  ],
  "Version": "1"
}
```

创建服务关联角色

管理员账号在具有服务管理权限的资源栈组中，创建资源栈实例时，在管理员账号下创建服务关联角色AliyunServiceRoleForROSStackGroupsRDAdmin，以此身份获取资源目录下的账号信息。在获取到的所有成员账号中创建服务关联角色AliyunServiceRoleForROSStackGroupsRDMember，以此身份动态创建 `stackgroups-exec-` 开头的普通RAM角色，并以此身份部署资源栈。 `stackgroups-exec-` 开头的角色会在资源栈实例被成功删除时删除。

更多信息，请参见步骤三：创建资源栈组。

删除服务关联角色

- 如果您需要删除管理员账号中的ROS服务关联角色（AliyunServiceRoleForROSStackGroupsRDAdmin），需要先删除管理员账号中所有具有服务管理权限的资源栈组。
  i. 删除资源栈组。具体操作，请参见删除资源栈组。
  ii. 删除服务关联角色。具体操作，请参见删除服务关联角色。
- 如果您需要删除成员账号中的ROS服务关联角色（AliyunServiceRoleForROSStackGroupsRDMember），

需要先删除与此成员账号相关的资源栈实例。

i. 删除资源栈实例。具体操作，请参见单个删除资源栈实例。

ii. 删除服务关联角色。具体操作，请参见删除服务关联角色。

4.4. 基础操作

4.4.1. 管理资源栈组

本文为您介绍如何查看、更新和删除资源栈组。

查看资源栈组

您可以查看资源栈组信息，包括资源栈组的基本信息、权限、实例和模板等。

1. 登录资源编排控制台。
2. 在左侧导航栏，单击**资源栈组**。
3. 在页面左上角的地域下拉列表，选择资源栈组所在的地域。
4. 在**资源栈组列表**页面，单击目标资源栈组名称。
5. 在资源栈组管理页面，查看资源栈组详情。
   - 单击**详细信息**页签，查看**基本信息**、**权限**和**部署配置**。

     > **说明** 仅具有服务管理权限的资源栈组支持查看部署配置。

   - 单击**实例**页签，查看目标账号下的资源栈实例。
   - 单击**操作**页签，查看资源栈组的操作状态。您可以单击操作ID，查看目标地域中目标账号的操作状态。
   - 单击**参数**页签，查看创建资源栈组时模板中指定的参数，包括ROS提供的以 `ALIYUN::` 开始的伪参数。
   - 单击**模板**页签，查看资源栈组对应的模板信息。

更新资源栈组

您可以根据需要，更新资源栈组的模板、模板参数、权限类型、部署目标等信息，以便更新资源栈组中的资源栈实例。

1. 在左侧导航栏，单击**资源栈组**。
2. 在页面左上角的地域下拉列表，选择资源栈组所在的地域。
3. 在**资源栈组列表**页面，单击目标资源栈组**操作**列的**更新**。
4. 在**选择模板**页面，选择是否更新模板，然后单击**下一步**。
   - **使用当前模板**：不更新模板。
   - **替换当前模板**：请输入更新后的模板。
5. 在**配置模板参数**页面，更新**资源栈组描述**和参数配置，然后单击**下一步**。
6. 在**配置资源栈组**页面，根据需求选择**自助管理权限**或**服务管理权限**，然后单击**下一步**。

   > **说明** 如果当前资源栈组下已有资源栈实例，不允许更换权限类型。

7. 在**设置部署选项**页面，更新相关参数，然后单击**下一步**。

请根据不同权限类型更新以下参数：

- **自助管理权限**：更新**账号**、**地域**、**资源组**、**最大并发账号数**、**容错**和**区域并发**。
  关于参数的更多信息， 请参见自助管理权限参数说明。
- **服务管理权限**：更新**资源目录**、**自动部署**、**账号删除行为**、**地域**、**资源组**、**最大并发账号数**、**容错**和**区域并发**。
  关于参数的更多信息， 请参见服务管理权限参数说明。

8. 在**检查并确认**页面，检查资源栈组信息无误后，单击**确认修改**。

删除资源栈组

删除资源栈组前，请确保您已经删除资源栈组中所有资源栈实例。具体操作，请参见单个删除资源栈实例。

1. 在左侧导航栏，单击**资源栈组**。
2. 在页面左上角的地域下拉列表，选择资源栈组所在的地域。
3. 在**资源栈组列表**页面，单击目标资源栈组**操作**列的**删除**。
4. 单击**确定**。

4.4.2. 管理资源栈实例

资源栈组创建完成后，您可以对资源栈实例进行更新或者删除操作。本文为您介绍如何更新和删除资源栈实例。

更新资源栈实例

1. 登录资源编排控制台。
2. 在左侧导航栏，单击**资源栈组**。
3. 在页面左上角的地域下拉列表，选择资源栈的所在地域。
4. 在**资源栈组列表**中，在目标资源栈组右侧**操作**列选择 **> 更新资源栈实例**。

5. 在**设置部署选项**页面，配置参数。
   - **自助管理权限**：关于参数配置的更多信息， 请参见自助管理权限参数说明。
   - **服务管理权限**：关于参数配置的更多信息， 请参见服务管理权限参数说明。
6. 在**指定覆盖**页面，选择目标参数，使用以下两种方式修改参数值，然后单击**下一步**。
   - 方式一：选择**编辑覆盖值 > 覆盖资源栈组参数值**对参数进行重置。
   - 方式二：选择**编辑覆盖值 > 重置覆盖字段**输入新的参数值。

   > **说明** 资源栈实例创建成功后，您可以在目标账号（账号B）的**资源栈列表**，单击资源栈ID，然后在**参数**页签下，查看重新指定的参数值。

7. 在**检查并确认**页面，单击**确认覆盖**。

单个删除资源栈实例

> **说明** 以下操作仅适用于具有自助管理权限的资源栈组关联的资源栈实例。

1. 登录资源编排控制台。
2. 在左侧导航栏，单击**资源栈组**。

3. 在页面左上角的地域下拉列表，选择资源栈的所在地域。
4. 在**资源栈组列表**中，单击**资源栈组名称**。
5. 选择**实例**页签，在目标资源栈实例所在行，单击**删除**。
6. 在弹出的对话框中，选择**删除方式**，并单击**确定**。
   - **保留资源**：只删除资源栈实例，不删除相应的资源栈。
   - **释放资源**：删除资源栈实例和相应的资源栈，请您谨慎操作。

批量删除资源栈实例

1. 登录资源编排控制台。
2. 在左侧导航栏，单击**资源栈组**。
3. 在页面左上角的地域下拉列表，选择资源栈的所在地域。
4. 在**资源栈组列表**中，在目标资源栈组右侧**操作**列选择 > **删除资源栈实例**。
5. 在**设置部署选项**页面，配置参数。
   - **自助管理权限**：关于参数配置的更多信息， 请参见自助管理权限参数说明。
   - **服务管理权限**：关于参数配置的更多信息， 请参见服务管理权限参数说明。
6. 在**检查并确认**页面，单击**确认删除**。
7. 选择资源栈实例删除方式，然后单击**确定**。
   - **保留资源**：只删除资源栈实例，不删除相应的资源栈。
   - **释放资源**：删除资源栈实例和相应的资源栈，请您谨慎操作。

4.4.3. 自动部署资源栈实例

当您开通了资源目录，使用资源目录的企业管理账号或委派管理员账号创建服务管理权限模式的资源栈组时，可以设置自动部署功能。此时当目标资源夹中成员账号变动时，可以自动部署资源栈实例，方便您统一管理成员账号中的资源栈实例。您也可以根据需要变更自动部署设置。

前提条件

请确保您已经创建了服务管理权限模式的资源栈组。具体操作，请参见步骤三：创建资源栈组。

应用场景

启用自动部署后，如果目标资源夹中新增了成员账号，资源栈组会自动将资源栈实例部署到该成员账号。如果目标资源夹中删除了成员账号，则资源栈组会自动删除该成员账号中的资源栈实例。

自动部署应用场景如下：

- 场景一：成员账号在资源夹之间移动时进行自动部署。
  假设资源栈组已对资源目录中的资源夹1（Folder1）和资源夹2（Folder2）设置了自动部署，当成员账号从资源夹1（Folder1）移动至资源夹2（Folder2）时，会触发自动部署。ROS将从资源夹1（Folder1）中成员账号的目标地域中删除资源栈实例，同时创建相同的资源栈实例，添加到资源夹2（Folder2）成员账号的目标地域中。
- 场景二：资源夹中新增成员账号时进行自动部署。
  假设资源栈组已对资源夹设置了自动部署，当资源夹中新增成员账号A（AccountA）时会触发自动部署，在成员账号A（AccountA）的目标地域中创建资源栈实例 。如果实例创建过程中该资源夹新增了另一个成员账号（AccountB），资源栈组会先在成员账号A（AccountA）中创建资源栈实例，创建完成后继续在成员账号（AccountB）的目标地域中创建资源栈实例。

> **说明** 您只能为资源栈组设置自动部署，不能为资源夹、成员账号或地域设置自动部署。

操作步骤

1. 使用资源目录的企业管理账号或委派管理员账号登录资源编排控制台。

   > **说明** 登录账号需要与创建资源栈组的账号相同。

2. 在左侧导航栏，单击**资源栈组**。
3. 在页面左上角的地域下拉列表，选择资源栈组所在的地域。
4. 单击目标资源栈组名称。
5. 在**详细信息**页签的**部署配置**区域，单击**编辑自动部署**。
6. 在**编辑自动部署**对话框，设置**自动部署**为**已启用**，然后设置**账号删除行为**。

   关于自动部署和账号删除行为的更多信息，请参见参数说明。

   > **注意** **账号删除行为**设置为**保留资源栈**时，资源栈实例将从资源栈组中移除，但资源栈及其相关资源将保留。资源保持当前状态，但不再是资源栈组的一部分。资源栈不能与现有资源栈组或新资源栈组重新关联。

7. 单击**保存**。

4.5. 资源栈组和资源栈实例状态码

资源编排服务ROS（Resource Orchestration Service）为资源栈组操作和资源栈实例生成状态代码。

资源栈组操作

资源栈组操作状态	描述
RUNNING	操作正在进行中。
SUCCEEDED	操作已完成，未超出操作的容错能力。
FAILED	操作失败的资源栈的数量超出了容错能力。
STOPPING	操作正在停止。
STOPPED	操作已停止。
QUEUED	操作排队中。对于具有服务管理权限的资源栈组，若自动部署时有操作在执行，由自动部署触发的操作会排队等待。更多信息，请参见自动部署资源栈实例。

资源栈实例操作

资源栈实例操作状态	描述
CURRENT	资源栈是资源栈组中最新的，即资源栈的模板、参数与资源栈组一致。

资源栈实例操作状态	描述
OUTDATED	资源栈不是资源栈组最新的，即资源栈的模板、参数与资源栈组不一致。例如： • 更新资源栈组及部分资源栈实例，未更新的资源栈实例状态为OUTDATED。 • 创建或更新资源栈实例时，角色扮演失败，创建或更新资源栈未执行，资源栈实例状态为OUTDATED。 • 创建或更新资源栈实例时，角色扮演成功，创建或更新资源栈失败，资源栈实例状态为OUTDATED。

5.资源场景

5.1. 概览

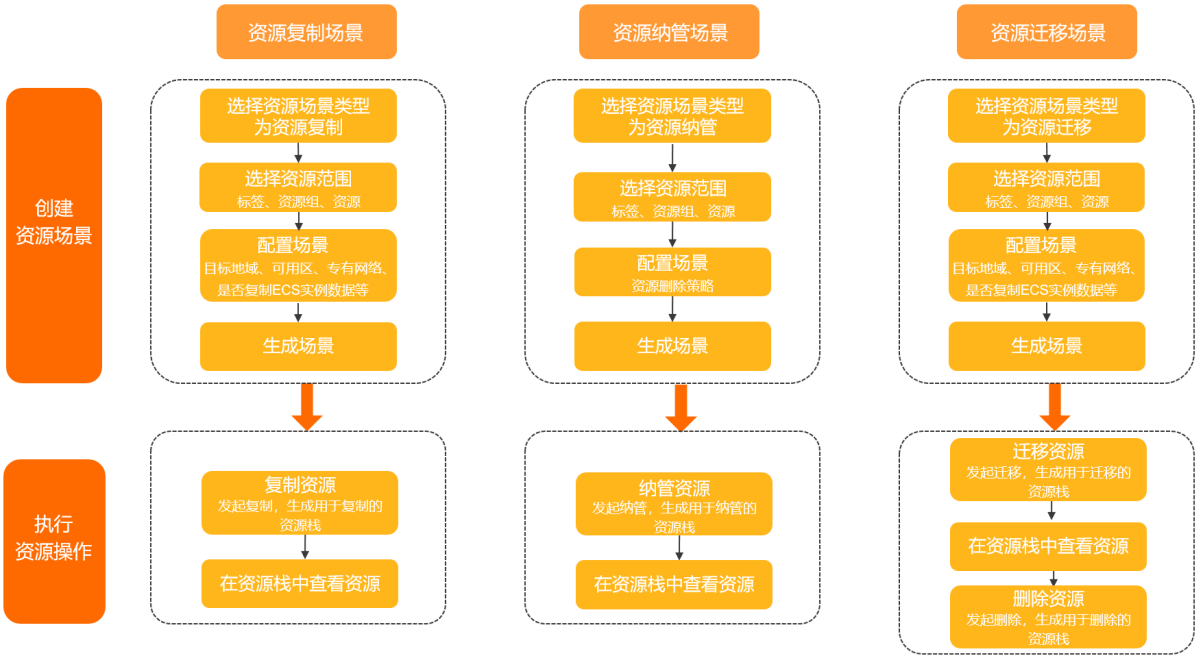
通过资源场景功能，您可以在可视化界面上选择资源范围，并对一组资源进行复制、纳管、迁移等操作，从而简化资源管理。

基本概念

概念	说明
资源复制	如果您需要复制一组资源及其依赖关系，可以创建资源复制类型的资源场景。该资源类型支持复制指定范围内的所有资源，生成一套架构完全相同的资源。 更多信息，请参见资源复制场景。
资源纳管	如果您需要将一组现有资源导入到一个新的资源栈中进行统一管理，可以创建资源纳管类型的资源场景。 更多信息，请参见资源纳管场景。
资源迁移	如果您需要迁移一组资源及其依赖关系，可以创建资源迁移类型的资源场景，在资源场景中迁移资源、生成资源栈，通过资源栈查看迁移进度。待迁移完成后，您还可以删除源资源。 更多信息，请参见资源迁移场景。
源节点	创建资源场景后，ROS会筛选指定范围内的所有资源，生成资源架构和属性信息，这些资源被称为源节点。 • 在资源复制类型的资源场景中，源节点用于存储已有资源的ID、类型、属性等信息。 • 在资源纳管类型的资源场景中，源节点中的资源将直接以资源导入的方式创建资源栈，从而纳管一组资源。
新节点	创建资源复制类型的资源场景后，ROS会基于源节点内容额外生成待复制的资源，包括其资源类型、属性等信息，这些资源被称为新节点。 新节点的资源属性和数量可能会和源节点有所不同，ROS在生成场景时会为新节点生成可用的规格、网段等属性，并会根据资源间的关联关系和资源特性决策生成出的资源类型和数量。 • 示例1：假设源节点的ECS实例规格为ecs.sn1ne.large（2核4 GB），在生成场景时无此规格库存，则会查找相似的实例规格并应用于新节点。 • 示例2：假设源节点仅包含EIP及其绑定资源，意味着EIP已绑定至某个实例。由于实例只能绑定1个EIP，在生成场景时新节点中只会包含EIP资源，而不包含EIP绑定的资源。

工作原理

ROS资源场景工作原理如下图所示。



您可以选择不同的资源场景类型，并通过标签、资源组或资源（资源类型和ID）三种不同方式选择资源范围，从而创建资源场景。ROS会筛选指定范围内的所有资源，生成资源架构和属性信息，即源节点。

- 资源复制场景
  ROS会额外生成待复制的资源，包括其资源类型、属性等信息，即新节点。创建资源场景成功后，即可复制资源。ROS会根据新节点的内容创建资源栈，从而复制出和源节点相同架构的一组资源。您可以在资源场景管理页面的资源栈页签中查看已创建的资源栈。
- 资源纳管场景
  创建资源场景成功后，即可纳管资源。ROS会根据源节点的内容以资源导入的方式创建资源栈，从而纳管一组资源。您可以在资源场景管理页面的资源栈页签中查看已创建的资源栈。
- 资源迁移场景
  ROS会额外生成待迁移的资源，包括其资源类型、属性等信息，即新节点。创建资源场景成功后，即可迁移资源。ROS会根据新节点的内容创建资源栈，从而复制出和源节点相同架构的一组资源。您可以在资源场景管理页面的资源栈页签中查看已创建的资源栈。迁移完成后，您可以选择删除源资源，ROS会将源资源纳管到新的资源栈中，再进行删除。

支持资源场景的资源类型

资源场景功能支持特定的资源类型。更多信息，请参见支持资源场景的资源类型。

5.2. 资源复制场景

如果您需要快速复制一组资源及其依赖关系，可以通过创建资源复制场景来实现。

背景信息

创建资源复制场景时，ROS会筛选指定范围内的所有资源并生成源节点，同时额外生成新节点，用于复制资源。ROS会根据新节点的内容创建资源栈，从而复制出和源节点相同架构的一组资源。您可以在资源场景管理页面的资源栈页签中查看已创建的资源栈。

更多信息，请参见基本概念和工作原理。

步骤一：创建资源场景

1. 登录资源编排控制台。

2. 在左侧导航栏，单击**资源场景**。
3. 在页面左上角的地域下拉列表，选择资源场景所在的地域。
4. 在**资源场景列表**页面，单击**创建资源场景**。
5. 在**创建资源场景**对话框，设置资源场景描述，然后选择资源场景为**资源复制**。
6. 选择资源范围。
   - 选择绑定指定标签的资源

     a. 在**选择方式**区域，选择**源标签**。
     b. 在**源标签**区域，设置**标签键**，然后根据需要设置**标签值**。
        最多支持设置10个标签。
     c. （可选）在**资源类型筛选**区域，选择**资源类型**。
        最多支持选择20个资源类型。
   - 选择指定资源组的资源

     a. 在**选择方式**区域，选择**源资源组**。
     b. 在**源资源组**区域，选择资源组。
     c. （可选）在**资源类型筛选**区域，选择**资源类型**。
        最多支持选择20个资源类型。
   - 选择指定资源

     a. 在**选择方式**区域，选择**源资源**。

b. 在**源资源**区域，选择**资源类型**和**资源ID**。
最多支持选择20个资源类型。

7. 配置资源场景，设置资源复制的目标参数。
   i. 设置**资源删除策略**。
      - **保留**：将资源复制至资源栈后，在删除已创建的资源栈时，保留原有资源。
      - **删除**：将资源复制至资源栈后，在删除已创建的资源栈时，默认删除原有资源；若在删除确认框中选择保留部分资源，则删除未选择保留的资源。
   ii. 设置**目标地域**。
   资源复制的目标地域，默认为当前地域。
   iii. 设置**目标可用区**。
   资源复制的目标可用区，在同地域下默认与源资源的可用区相同；若跨地域，默认由系统自动选择。
   iv. 设置**目标专有网络**。
   资源复制的目标专有网络，在同地域下默认与源资源的专有网络相同；若跨地域，默认由系统自动创建。

   > **说明** 请确保目标专有网络的CIDR包含源资源所属专有网络的CIDR。

   v. 设置**目标交换机**。
   资源复制的目标交换机，在同地域下默认与源资源的交换机相同；若跨地域或者指定了目标专有网络，默认由系统自动创建。

   > **说明** 请确保目标交换机的CIDR包含源资源所属交换机的CIDR。

   vi. 设置**是否复制ECS实例数据**。
   当源资源里包含ECS实例时，该参数有效，表示复制ECS实例数据。为了确保数据一致性，请您停止源实例后再进行复制。
8. 单击**生成场景**。
   创建资源场景成功后，**状态**列显示**生成完成**。

步骤二：复制资源

1. 在**资源场景列表**页面，单击目标资源场景ID。
2. 在资源场景管理页面，单击右上角的**复制资源**。
3. 单击**确定复制**。
4. 复制资源成功后，单击**关闭**。

步骤三：查看资源

1. 在资源场景管理页面，单击**资源栈**页签，然后单击目标资源栈ID。
2. 在资源栈管理页面，单击**资源**页签，查看复制后的资源详情。

5.3. 资源纳管场景

如果您需要将一组现有资源导入到同一资源栈中进行统一管理，可以通过创建资源纳管场景来实现。

背景信息

创建资源纳管类型的资源场景时，ROS会筛选指定范围内的所有资源，生成资源架构和属性信息。您可以将这些资源以资源导入的方式创建资源栈，从而纳管一组资源，然后在资源场景管理页面的资源栈页签中查看已创建的资源栈。

关于资源纳管类型资源场景的更多信息，请参见工作原理。

步骤一：创建资源场景

1. 登录资源编排控制台。
2. 在左侧导航栏，单击**资源场景**。
3. 在页面左上角的地域下拉列表，选择资源场景所在的地域。
4. 在**资源场景列表**页面，单击**创建资源场景**。
5. 在**创建资源场景**对话框，设置资源场景描述，然后选择资源场景为**资源纳管**。
6. 选择资源范围。
   - 选择绑定指定标签的资源

     a. 在**选择方式**区域，选择**源标签**。
     b. 在**源标签**区域，设置**标签键**，然后根据需要设置**标签值**。
        最多支持设置10个标签。
     c. （可选）在**资源类型筛选**区域，选择**资源类型**。
        最多支持选择20个资源类型。
   - 选择指定资源组的资源

     a. 在**选择方式**区域，选择**源资源组**。
     b. 在**源资源组**区域，选择资源组。
     c. （可选）在**资源类型筛选**区域，选择**资源类型**。
        最多支持选择20个资源类型。
   - 选择指定资源

a. 在**选择方式**区域，选择**源资源**。

b. 在**源资源**区域，选择**资源类型**和**资源ID**。
最多支持选择20个资源类型。

7. 设置资源场景的**资源删除策略**。
   - **保留**：将资源纳管至资源栈后，在删除已创建的资源栈时，保留原有资源。
   - **删除**：将资源纳管至资源栈后，在删除已创建的资源栈时，默认删除原有资源；若在删除确认框中选择保留部分资源，则删除未选择保留的资源。
8. 单击**生成场景**。

   创建资源场景成功后，**状态**列显示**生成完成**。

步骤二：纳管资源

1. 在**资源场景列表**页面，单击目标资源场景ID。
2. 在资源场景管理页面，单击右上角的**纳管资源**。
3. 单击**确定纳管**。
4. 资源纳管成功后，单击**关闭**。

步骤三：查看资源

1. 在资源场景管理页面，单击**资源栈**页签，然后单击目标资源栈ID。
2. 在资源栈管理页面，单击**资源**页签，查看纳管后的资源详情。

5.4. 资源迁移场景

如果您需要迁移一组资源及其依赖关系，可以通过创建资源迁移场景来实现。

背景信息

创建资源迁移场景时，ROS会筛选指定范围内的所有资源并生成源节点，同时额外生成新节点，用于迁移资源。发起迁移后，ROS会根据新节点的内容创建资源栈，从而复制出和源节点相同架构的一组资源。迁移完成后，您可以选择删除源资源，ROS会将源资源纳管到新的资源栈中，再进行删除。

您可以在资源场景管理页面的资源栈页签中查看已创建的资源栈。更多信息，请参见基本概念和工作原理。

步骤一：创建资源场景

1. 登录资源编排控制台。
2. 在左侧导航栏，单击**资源场景**。
3. 在页面左上角的地域下拉列表，选择资源场景所在的地域。
4. 在**资源场景列表**页面，单击**创建资源场景**。

5. 在**创建资源场景**对话框，设置资源场景描述，然后选择资源场景为**资源迁移**。
6. 选择资源范围。
   - 选择绑定指定标签的资源

     a. 在**选择方式**区域，选择**源标签**。
     b. 在**源标签**区域，设置**标签键**，然后根据需要设置**标签值**。
        最多支持设置10个标签。
     c. （可选）在**资源类型筛选**区域，选择**资源类型**。
        最多支持选择20个资源类型。
   - 选择指定资源组的资源

     a. 在**选择方式**区域，选择**源资源组**。
     b. 在**源资源组**区域，选择资源组。
     c. （可选）在**资源类型筛选**区域，选择**资源类型**。
        最多支持选择20个资源类型。
   - 选择指定资源

     a. 在**选择方式**区域，选择**源资源**。
     b. 在**源资源**区域，选择**资源类型**和**资源ID**。
        最多支持选择20个资源类型。
7. 配置资源场景，设置资源迁移的目标参数。

i. 设置**目标地域**。

资源迁移的目标地域，默认为当前地域。

ii. 设置**目标可用区**。

资源迁移的目标可用区，在同地域下默认与源资源的可用区相同；若跨地域，默认由系统自动选择。

iii. 设置**目标专有网络**。

资源迁移的目标专有网络，在同地域下默认与源资源的专有网络相同；若跨地域，默认由系统自动创建。

> **说明** 请确保目标专有网络的CIDR包含源资源所属专有网络的CIDR。

iv. 设置**目标交换机**。

资源迁移的目标交换机，在同地域下默认与源资源的交换机相同；若跨地域或者指定了目标专有网络，默认由系统自动创建。

> **说明** 请确保目标交换机的CIDR包含源资源所属交换机的CIDR。

v. 设置**是否复制ECS实例数据**。

当源资源里包含ECS实例时，该参数有效，表示复制ECS实例数据。为了确保数据一致性，请您停止源实例后再进行复制。

8. 单击**生成场景**。

创建资源场景成功后，**状态**列显示**生成完成**。

步骤二：迁移资源

1. 在**资源场景列表**页面，单击目标资源场景ID。
2. 在资源场景管理页面，单击右上角的**迁移资源**。
3. 单击**确定迁移**。
4. 资源迁移成功后，单击**关闭**。

步骤三：查看资源

1. 在资源场景管理页面，单击**资源栈**页签，然后单击目标资源栈ID。
2. 在资源栈管理页面，单击**资源**页签，查看迁移后的资源详情。

步骤四（可选）：删除源资源

1. 在资源场景管理页面，单击右上角的**删除源资源**。
2. 单击**确定删除**。

> **说明** 删除资源前会先将所有资源纳管至新的资源栈中。

3. 删除源资源成功后，单击**关闭**。

相关视频

5.5. 支持资源场景的资源类型

本文为您介绍支持资源场景的资源类型，以及这些资源类型支持的资源范围选择方式。

云服务	资源类型	附属资源类型	资源范围选择方式		
			源标签	源资源组	源资源
云服务器ECS	ALIYUN::ECS::Disk	ALIYUN::ECS::DiskAttachment	□	□	□
	ALIYUN::ECS::Instance	无	□	□	□
	ALIYUN::ECS::NetworkInterface	ALIYUN::ECS::NetworkInterfaceAttachment	□	□	□
	ALIYUN::ECS::SecurityGroup	无	□	□	□
	ALIYUN::ECS::VPC	无	□	□	□
	ALIYUN::ECS::VSwitch	无	□	□	□
对象存储OSS	ALIYUN::OSS::Bucket	无	□	□	□
云数据库PolarDB	ALIYUN::POLARDB::DBCluster	无	□	□	□
访问控制RAM	ALIYUN::RAM::Group	无	□	□	□
	ALIYUN::RAM::ManagedPolicy	无	□	□	□
	ALIYUN::RAM::Role	无	□	□	□
	ALIYUN::RAM::User	无	□	□	□
云数据库RDS	ALIYUN::RDS::DBInstance	无	□	□	□
云数据库Redis版	ALIYUN::REDIS::Instance	ALIYUN::REDIS::Whitelist	□	□	□
负载均衡SLB	ALIYUN::SLB::LoadBalancer	• ALIYUN::SLB::Listener • ALIYUN::SLB::VServerGroup • ALIYUN::SLB::BackendServerAttachment • ALIYUN::SLB::MasterSlaveServerGroup	□	□	□
专有网络VPC	ALIYUN::VPC::EIP	ALIYUN::VPC::EIPAssociation	□	□	□

5.6. 基础操作

5.6.1. 创建资源场景

通过资源场景功能，您可以在可视化界面上选择资源范围，并对一组资源进行复制、纳管、迁移等操作，从而简化资源管理。

创建资源复制场景

1. 登录资源编排控制台。
2. 在左侧导航栏，单击**资源场景**。
3. 在页面左上角的地域下拉列表，选择资源场景所在的地域。
4. 在**资源场景列表**页面，单击**创建资源场景**。
5. 在**创建资源场景**对话框，设置资源场景描述，然后选择资源场景为**资源复制**。
6. 选择资源范围。
   - 选择绑定指定标签的资源

     a. 在**选择方式**区域，选择**源标签**。
     b. 在**源标签**区域，设置**标签键**，然后根据需要设置**标签值**。
        最多支持设置10个标签。
     c. （可选）在**资源类型筛选**区域，选择**资源类型**。
        最多支持选择20个资源类型。
   - 选择指定资源组的资源

     a. 在**选择方式**区域，选择**源资源组**。
     b. 在**源资源组**区域，选择资源组。
     c. （可选）在**资源类型筛选**区域，选择**资源类型**。
        最多支持选择20个资源类型。
   - 选择指定资源

a. 在**选择方式**区域，选择**源资源**。

b. 在**源资源**区域，选择**资源类型**和**资源ID**。
最多支持选择20个资源类型。

7. 配置资源场景，设置资源复制的目标参数。

i. 设置**资源删除策略**。

- **保留**：将资源复制至资源栈后，在删除已创建的资源栈时，保留原有资源。
- **删除**：将资源复制至资源栈后，在删除已创建的资源栈时，默认删除原有资源；若在删除确认框中选择保留部分资源，则删除未选择保留的资源。

ii. 设置**目标地域**。

资源复制的目标地域，默认为当前地域。

iii. 设置**目标可用区**。

资源复制的目标可用区，在同地域下默认与源资源的可用区相同；若跨地域，默认由系统自动选择。

iv. 设置**目标专有网络**。

资源复制的目标专有网络，在同地域下默认与源资源的专有网络相同；若跨地域，默认由系统自动创建。

> **说明** 请确保目标专有网络的CIDR包含源资源所属专有网络的CIDR。

v. 设置**目标交换机**。

资源复制的目标交换机，在同地域下默认与源资源的交换机相同；若跨地域或者指定了目标专有网络，默认由系统自动创建。

> **说明** 请确保目标交换机的CIDR包含源资源所属交换机的CIDR。

vi. 设置**是否复制ECS实例数据**。

当源资源里包含ECS实例时，该参数有效，表示复制ECS实例数据。为了确保数据一致性，请您停止源实例后再进行复制。

8. 单击**生成场景**。

创建资源场景成功后，**状态**列显示**生成完成**。

创建资源纳管场景

1. 登录资源编排控制台。
2. 在左侧导航栏，单击**资源场景**。
3. 在页面左上角的地域下拉列表，选择资源场景所在的地域。

4. 在**资源场景列表**页面，单击**创建资源场景**。
5. 在**创建资源场景**对话框，设置资源场景描述，然后选择资源场景为**资源纳管**。
6. 选择资源范围。
   - 选择绑定指定标签的资源

     a. 在**选择方式**区域，选择**源标签**。
     b. 在**源标签**区域，设置**标签键**，然后根据需要设置**标签值**。
        最多支持设置10个标签。
     c. （可选）在**资源类型筛选**区域，选择**资源类型**。
        最多支持选择20个资源类型。
   - 选择指定资源组的资源

     a. 在**选择方式**区域，选择**源资源组**。
     b. 在**源资源组**区域，选择资源组。
     c. （可选）在**资源类型筛选**区域，选择**资源类型**。
        最多支持选择20个资源类型。
   - 选择指定资源

     a. 在**选择方式**区域，选择**源资源**。
     b. 在**源资源**区域，选择**资源类型**和**资源ID**。
        最多支持选择20个资源类型。

7. 设置资源场景的**资源删除策略**。
   - **保留**：将资源纳管至资源栈后，在删除已创建的资源栈时，保留原有资源。
   - **删除**：将资源纳管至资源栈后，在删除已创建的资源栈时，默认删除原有资源；若在删除确认框中选择保留部分资源，则删除未选择保留的资源。
8. 单击**生成场景**。

   创建资源场景成功后，**状态**列显示**生成完成**。

创建资源迁移场景

1. 登录资源编排控制台。
2. 在左侧导航栏，单击**资源场景**。
3. 在页面左上角的地域下拉列表，选择资源场景所在的地域。
4. 在**资源场景列表**页面，单击**创建资源场景**。
5. 在**创建资源场景**对话框，设置资源场景描述，然后选择资源场景为**资源迁移**。
6. 选择资源范围。
   - 选择绑定指定标签的资源

     a. 在**选择方式**区域，选择**源标签**。

     b. 在**源标签**区域，设置**标签键**，然后根据需要设置**标签值**。
     最多支持设置10个标签。

     c. （可选）在**资源类型筛选**区域，选择**资源类型**。
     最多支持选择20个资源类型。

   - 选择指定资源组的资源

     a. 在**选择方式**区域，选择**源资源组**。

     b. 在**源资源组**区域，选择资源组。

     c. （可选）在**资源类型筛选**区域，选择**资源类型**。
     最多支持选择20个资源类型。

   - 选择指定资源

a. 在**选择方式**区域，选择**源资源**。

b. 在**源资源**区域，选择**资源类型**和**资源ID**。
最多支持选择20个资源类型。

7. 配置资源场景，设置资源迁移的目标参数。

i. 设置**目标地域**。

资源迁移的目标地域，默认为当前地域。

ii. 设置**目标可用区**。

资源迁移的目标可用区，在同地域下默认与源资源的可用区相同；若跨地域，默认由系统自动选择。

iii. 设置**目标专有网络**。

资源迁移的目标专有网络，在同地域下默认与源资源的专有网络相同；若跨地域，默认由系统自动创建。

> **说明** 请确保目标专有网络的CIDR包含源资源所属专有网络的CIDR。

iv. 设置**目标交换机**。

资源迁移的目标交换机，在同地域下默认与源资源的交换机相同；若跨地域或者指定了目标专有网络，默认由系统自动创建。

> **说明** 请确保目标交换机的CIDR包含源资源所属交换机的CIDR。

v. 设置**是否复制ECS实例数据**。

当源资源里包含ECS实例时，该参数有效，表示复制ECS实例数据。为了确保数据一致性，请您停止源实例后再进行复制。

8. 单击**生成场景**。

创建资源场景成功后，**状态**列显示**生成完成**。

5.6.2. 查看资源场景

本文为您介绍如何查看资源场景详情，包括资源场景的基本信息、资源架构图、源节点、新节点和关联资源栈等。

前提条件

请确保您已经创建资源场景。具体操作，请参见创建资源场景。

操作步骤

1. 登录资源编排控制台。

2. 在左侧导航栏，单击**资源场景**。
3. 在页面左上角的地域下拉列表，选择资源场景所在的地域。
4. 在**资源场景列表**页面，单击目标资源场景ID。
5. 在资源场景管理页面，查看资源场景详情。
   i. 单击**资源场景详情**页签，查看**基本信息**和**资源架构图**。
   ii. 在**资源场景详情**页签，查看资源详情。
   - 资源复制场景
     - 单击**源节点**页签，查看资源场景指定的资源范围所扫描出的资源列表。然后单击目标资源**属性**列的**查看**，查看资源详情。
     - 单击**新节点**页签，查看资源场景待创建的资源列表。然后单击目标资源**属性**列的**查看**，查看资源详情。
   - 资源纳管场景
     在页面右下角，查看资源场景指定的资源范围所扫描出的资源列表。然后单击目标资源**属性**列的**查看**，查看资源详情。
   - 资源迁移场景
     - 单击**源节点**页签，查看资源场景指定的资源范围所扫描出的资源列表。然后单击目标资源**属性**列的**查看**，查看资源详情。
     - 单击**新节点**页签，查看资源场景待迁移的资源列表。然后单击目标资源**属性**列的**查看**，查看资源详情。

   iii. 单击**资源栈**页签，查看资源场景创建的资源栈列表。

5.6.3. 修改资源场景

当资源不存在时，可能出现创建资源场景失败的情况，此时您可以修改资源场景，重新选择正确的资源。

前提条件

请确保您已经创建资源场景，且资源场景状态为**生成失败**。具体操作，请参见创建资源场景。

背景信息

资源场景类型与您创建资源场景时所选的类型相同，不支持修改。

操作步骤

1. 登录资源编排控制台。
2. 在左侧导航栏，单击**资源场景**。
3. 在页面左上角的地域下拉列表，选择资源场景所在的地域。
4. 在**资源场景列表**页面，单击目标资源场景**操作**列的**修改**。
5. 在**修改资源场景**对话框，重新选择资源范围。
   - 选择绑定指定标签的资源

a. 在**选择方式**区域，选择**源标签**。

b. 在**源标签**区域，设置**标签键**，然后根据需要设置**标签值**。
最多支持设置10个标签。

c. （可选）在**资源类型筛选**区域，选择**资源类型**。
最多支持选择20个资源类型。

- 选择指定资源组的资源

a. 在**选择方式**区域，选择**源资源组**。

b. 在**源资源组**区域，选择资源组。

c. （可选）在**资源类型筛选**区域，选择**资源类型**。
最多支持选择20个资源类型。

- 选择指定资源

a. 在**选择方式**区域，选择**源资源**。

b. 在**源资源**区域，选择**资源类型**和**资源ID**。
最多支持选择20个资源类型。

6. 修改资源场景设置，然后单击**确认修改**。

执行结果

修改资源场景成功后，**状态**列显示**生成完成**。

5.6.4. 删除资源场景

本文为您介绍如何删除资源场景。

前提条件

- 请确保资源场景没有处于生成中。
- 如果您使用RAM用户删除资源场景，请确保该RAM用户具有AliyunROSFullAccess权限。具体操作，请参见为RAM用户授权。

操作步骤

1. 登录资源编排控制台。
2. 在左侧导航栏，单击**资源场景**。
3. 在页面左上角的地域下拉列表，选择资源场景所在的地域。
4. 在**资源场景列表**页面，单击目标资源场景**操作**列的**删除**。
5. 单击**确定**。

> **说明** 删除资源场景不会删除其关联的资源栈。如果需要删除资源栈，可以在资源栈列表页面中删除。具体操作，请参见删除资源栈。

5.6.5. 生成资源场景模板

如果您需要修改资源场景中部分资源，可以生成资源场景模板，然后基于此模板修改内容，从而创建资源栈并复制、纳管或迁移资源。

前提条件

请确保您已经创建资源场景。具体操作，请参见创建资源场景。

背景信息

您可以在不同类型的资源场景中生成和查看模板，并使用此模板复制、纳管或迁移资源：

- 资源复制场景
  生成新节点模板，即计划复制出的一组资源的模板。您可以基于此模板修改内容，然后使用模板手动创建资源栈以便复制资源。您也可以直接在资源复制场景中复制资源。具体操作，请参见创建资源栈和资源复制场景。
- 资源纳管源场景
  生成源节点模板，即指定资源范围内的一组资源的模板。您可以基于此模板修改内容，然后使用模板手动创建资源栈以便纳管资源。您也可以直接在资源纳管场景中纳管资源。具体操作，请参见创建资源栈和资源纳管场景。
- 资源迁移场景
  生成新节点模板，即计划迁移的一组资源的模板。您可以基于此模板修改内容，然后使用模板手动创建资源栈以便迁移资源。您也可以直接在资源迁移场景中复制资源。具体操作，请参见创建资源栈和资源迁移场景。

操作步骤

1. 登录资源编排控制台。
2. 在左侧导航栏，单击**资源场景**。
3. 在页面左上角的地域下拉列表，选择资源场景所在的地域。

4. 在**资源场景列表**页面，单击目标资源场景ID。
5. 在资源场景管理页面，单击右上角的**生成模板**。
6. 在**模板内容**对话框，获取资源场景对应的模板。
7. 单击**关闭**。

6.模板

6.1. 管理模板

6.1.1. 我的模板

6.1.1.1. 创建模板

模板用于创建资源栈，是描述基础设施和架构的蓝图。本文为您介绍如何创建模板。

操作步骤

1. 登录资源编排控制台。
2. 在左侧导航栏，选择**模板>我的模板**。
3. 在**我的模板**页面，单击**创建模板**。
4. 在弹出的**创建模板**对话框，输入**模板名称**和**模板描述**，然后选择**资源组**。

   > **说明** 如果不指定资源组，模板将加入默认资源组。关于如何创建资源组，请参见创建资源组。

5. 添加**标签**。

   关于如何添加标签，请参见创建标签。
6. 选择**模板录入方式**。
   - **使用url**：输入模板的下载地址，单击**获取JSON内容**，系统自动填入模板内容。
   - **输入模板**：手动输入模板内容。
   - **模板示例**：选择**模板示例**，系统自动填入模板内容。
7. 单击**确定**。

6.1.1.2. 编辑模板

本文为您介绍如何编辑模板。

> **说明**
> - 编辑模板后，模板版本加1。例如：版本由v1变为v2。
> - 模板最多支持设置100个版本。如果版本达到上限，模板编辑操作将失败，您需要重新创建模板。

操作步骤

1. 登录资源编排控制台。
2. 在左侧导航栏，选择**模板>我的模板**。
3. 在**我的模板**页面找到待编辑的模板，单击右侧**操作**列的**编辑**。
4. 在弹出的**修改模板**对话框，填写修改后的**模板名称**、**模板描述**。

5. 单击标签右侧的**添加**或**编辑**，添加或编辑标签。
6. 选择**模板录入方式**，根据需要修改模板内容。
   - **使用url**：输入模板的下载地址，单击**获取JSON内容**，系统自动填入模板内容。
   - **输入模板**：手动输入模板内容。
   - **模板示例**：选择**模板示例**，系统自动填入模板内容。
7. 单击**确定**。

   编辑模板后，您可以在**我的模板**页面模板**最新版本**列，查看模板的当前版本。

6.1.1.3. 删除模板

本文为您介绍如何删除模板。

前提条件

如果已经将模板共享给其他阿里云账号，需要取消共享后才能删除模板。具体操作，请参见取消共享。

操作步骤

1. 登录资源编排控制台。
2. 在左侧导航栏，选择**模板>我的模板**。
3. 在**我的模板**页面找到待删除的模板，单击右侧**操作**列的**删除**。
4. 在弹出的**删除**对话框，单击**确定**。

6.1.1.4. 通过模板创建资源栈

本文为您介绍如何通过已有模板创建资源栈。

前提条件

请确保您已创建了模板，操作方法请参见创建模板。

操作步骤

1. 登录资源编排控制台。
2. 在左侧导航栏，选择**模板>我的模板**。
3. 在**我的模板**页面找到待使用的模板，单击右侧操作列的**创建栈**。
4. 在**配置模板参数**页面，输入密码信息，单击**创建**。

6.1.2. 共享模板

6.1.2.1. 将模板共享给阿里云账号

您可以通过资源编排控制台添加共享，将已有模板共享给其他阿里云账号，以便统一管理模板，降低操作成本。本文以阿里云账号A将模板共享给阿里云账号B为例，为您进行介绍。

前提条件

- 请确保您已经创建模板。具体操作，请参见创建模板。
- 请先获取共享目标的阿里云账号ID。本文示例中阿里云账号B的ID为 `123435555956****` 。

背景信息

添加共享时，共享目标需要满足以下限制：

- 目标账号必须是阿里云账号，不能是RAM用户。
- 模板不能共享给自己。例如：您使用阿里云账号A登录，则不能将共享目标设置为阿里云账号A。
- 一个模板最多支持共享给50个阿里云账号。

操作步骤

1. 使用阿里云账号A登录资源编排控制台。
2. 在左侧导航栏，选择**模板>我的模板**。
3. 在**我的模板**页面，找到待添加共享的模板，单击**模板名称**。
4. 在**模板详情**区域，单击共享状态右侧的**编辑**。
5. 在**模板共享状态**对话框，单击**添加共享**。
6. 在**添加共享**对话框，输入**共享目标**（本文示例为：`123435555956****`）。

   > **说明** 您可以输入多个阿里云账号，多个账号间用半角逗号（,）间隔。

7. 选择共享版本。
   - **所有版本**：共享模板的所有版本。
   - **指定版本**：只共享模板指定的单个版本。此时您需要指定版本。
   - **始终保持最新版本**：只共享模板的最新版本。模板版本增加时，共享的版本也随之变化，始终保持最新版本。
8. 单击**确定**。

   添加共享成功后，共享目标（阿里云账号B）可以使用共享模板进行创建资源栈、创建资源栈组等操作。

6.1.2.2. 将模板共享给资源目录中的成员

在资源目录内，使用阿里云提供的资源共享功能，管理账号或成员可以将自己的ROS模板共享给其他成员使用，以便企业在特定的账号下统一管理模板，降低操作成本。

前提条件

请确保您已经启用资源目录组织共享功能。具体操作，请参见启用资源目录组织共享。

背景信息

您可以通过创建共享单元的方式共享ROS模板。

- 关于创建共享单元的限制，请参见使用限制。
- 您可以在创建共享单元时添加模板和模板的使用者，也可以先创建共享单元再添加。具体操作，请参见资源所有者添加或移除共享资源和资源所有者添加或移除资源使用者。

步骤一：在资源共享控制台共享模板

1. 登录资源共享控制台。
2. 在左侧导航栏，选择**资源共享 > 我的共享**。
3. 在页面左上角的地域下拉列表，选择ROS模板所在的地域。

   > **说明** 由于模板是全局资源，您只能选择华东2（上海）地域创建共享单元。

4. 单击**创建共享单元**。
5. 输入**共享单元名称**。
6. 在**选择共享的资源**区域，先选择**资源类型**为**资源编排（ROS）模板**，再选择**资源**（例如：template_vpc01），最后单击**添加**。
7. 在**添加资源使用者**区域，为ROS模板添加使用者。

   本文将以ROS模板共享给资源目录成员Alice为例进行介绍。

   i. 选择**添加方式**为**手动添加**。
   ii. 选择**使用者类型**为**阿里云账号**。
   iii. 输入**使用者ID**为成员Alice的ID。
   iv. 单击**添加**。
8. 单击**确定**。

   如果共享单元状态显示为**已启用**，表示共享单元创建成功。
   单击共享单元的ID链接，如果**共享的资源**和**资源使用者**的状态显示为**已关联**，表示共享的资源和资源使用者添加成功。

步骤二：在资源编排控制台使用共享模板

1. 访问成员Alice。

   具体操作，请参见访问成员。
2. 在资源编排控制台使用共享模板创建资源栈。

   i. 登录资源编排控制台。
   ii. 在左侧导航栏，单击**资源栈**。
   iii. 在页面左上角的地域下拉列表，选择资源栈的所在地域。
   iv. 在**资源栈列表**页面，单击**创建资源栈**，然后在下拉列表中选择**使用新资源（标准）**。
   v. 选择**模板录入方式**为**我的模板**，然后选择已共享的模板（例如：template_vpc01），最后单击**下一步**。
   vi. 单击**创建**。

6.1.2.3. 编辑共享

当您需要修改已共享模板的共享版本时，您可以编辑共享。

背景信息

编辑共享只能修改共享版本，不能修改共享目标。如果您需要修改共享目标，可以添加共享或取消共享。具体操作，请参见将模板共享给阿里云账号和取消共享。

操作步骤

1. 登录资源编排控制台。
2. 在左侧导航栏，选择**模板>我的模板**。
3. 在**我的模板**页面，找到待编辑共享的模板，单击**模板名称**。
4. 在**模板详情**区域，单击共享状态右侧的**编辑**。
5. 在弹出的**模板共享状态**对话框，找到待编辑共享的共享目标，单击右侧**操作**列的**编辑**。
6. 在弹出的**编辑共享**对话框，修改共享版本。

- **所有版本**：共享模板的所有版本。
- **指定版本**：只共享模板指定的单个版本。此时您需要指定版本。
- **始终保持最新版本**：只共享模板的最新版本。模板版本增加时，共享的版本也随之变化，始终保持最新版本。

7. 单击**确定**。

   修改共享成功后，您可以在**模板共享状态**对话框查看共享版本信息。

6.1.2.4. 取消共享

本文为您介绍如何为模板取消共享。

前提条件

请确保您已经为模板添加共享。具体操作，请参见将模板共享给阿里云账号。

操作步骤

1. 登录资源编排控制台。
2. 在左侧导航栏，选择**模板>我的模板**。
3. 在**我的模板**页面，找到待取消共享的模板，单击**模板名称**。
4. 在**模板详情**区域，单击共享状态右侧的**编辑**。
5. 在弹出的**模板共享状态**对话框，取消共享。
   - 为单个共享目标取消共享：找到待取消共享的共享目标，单击右侧**操作**列的**取消共享**。
   - 同时为多个共享目标取消共享：选中共享目标左侧的复选框，然后单击**取消共享**。
6. 在弹出的对话框，单击**确定**。

   取消共享后，当您使用共享目标的账号登录资源编排控制台时，将无法在**模板 > 共享模板**路径下查看共享模板，也无法再使用共享模板创建资源栈。

6.1.2.5. 查看您共享的模板

本文为您介绍如何查看您共享给其他阿里云账号的模板。

操作步骤

1. 登录资源编排控制台。
2. 在左侧导航栏，选择**模板>我的模板**。
3. 在**我的模板**页面，单击模板名称。
4. 单击**共享状态**右侧的**编辑**。
5. 在弹出的**模板共享状态**对话框，查看共享目标和共享版本信息。

6.1.2.6. 查看共享给您的模板

本文为您介绍如何查看其他阿里云账号共享给您的模板。

操作步骤

1. 登录资源编排控制台。
2. 在左侧导航栏，选择**模板>共享模板**。

3. 在**共享模板**页面，查看共享模板列表。
4. 单击共享模板右侧**操作**列的**详情**，查看模板ID、模板版本等信息。

> **说明** 如果添加共享时选择了**所有版本**，则可以查看不同版本的模板信息。

6.2. 模板语法

6.2.1. 模板结构说明

模板是一个JSON或YAML格式的文本文件，使用UTF-8编码。模板用于创建资源栈，是描述基础设施和架构的蓝图。模板编辑者在模板中定义阿里云资源和配置细节，并说明资源间的依赖关系。

ROS模板结构

```
{
  "ROSTemplateFormatVersion" : "2015-09-01",
  "Description" : "模板描述信息，可用于说明模板的适用场景、架构说明等。",
  "Metadata" : {
    // 关于模板的元数据信息，例如存放用于可视化的布局信息。
  },
  "Parameters" : {
    // 定义创建资源栈时，用户可以定制化的参数。
  },
  "Mappings" : {
    // 定义映射信息表，映射信息是一种多层的Map结构。
  },
  "Conditions": {
    // 使用内部条件函数定义条件。这些条件确定何时创建关联的资源。
  },
  "Resources" : {
    // 所需资源的详细定义，包括资源间的依赖关系、配置细节等。
  },
  "Outputs" : {
    // 用于输出一些资源属性等有用信息。可以通过API或控制台获取输出的内容。
  }
}
```

ROSTemplateFormatVersion（必选）

ROS支持的模板版本号，当前版本号：2015-09-01。

Description（可选）

模板的描述信息。可用于说明模板的适用场景、架构说明等。通常情况下，对模板进行详细描述，有利于用户理解模板的内容。

Metadata（可选）

模板编写者可以使用Metadata来存放与模板相关的元数据信息，内容可以为JSON格式。

Parameters（可选）

定义创建资源栈时，模板用户可以定制化的参数。通常，模板的编辑者会把ECS的规格设计成一个参数。参数支持默认值。使用参数可以增强模板的灵活性，提高复用性。使用模板创建资源栈时，可以根据实际的评估结果来选择合适的规格。

更多详细信息，请参见参数（Parameters）。

Mappings（可选）

Mappings定义了一个多层的映射表，可以通过Fn::FindInMap函数来选择Key对应的值，或根据不同的输入参数值作为Key来查找映射表。例如，您可以根据Region不同，自动查找Region-镜像映射表，从而找到适用的镜像。

更多详细信息，请参见映射（Mappings）。

Conditions（可选）

Conditions使用Fn::And、Fn::Or、Fn::Not、Fn::Equals定义条件。多个条件之间使用半角逗号（,）隔开。在创建或更新资源栈时，系统先计算模板中的所有条件，然后再创建资源。创建与true条件关联的所有资源，忽略与false条件关联的所有资源。

更多详细信息，请参见条件（Conditions）。

Resources（可选）

用于详细定义使用该模板创建的资源栈所包含的资源，包括资源间的依赖关系、配置细节等。

更多详细信息，请参见资源（Resources）。

Outputs（可选）

用于输出一些资源属性等有用信息。可以通过API或控制台获取输出的内容。

更多详细信息，请参见输出（Outputs）。

6.2.2. 参数（Parameters）

6.2.2.1. 概览

在创建模板时，使用参数（Parameters）可提高模板的灵活性和可复用性。创建资源栈时，可根据实际情况，替换模板中的某些参数值。

语法

每个参数由参数名称和参数属性组成。参数名称必须为英文字母、数字，并且在同一个模板中不能与其它参数名称重复。可以用Label字段来定义参数别名。

参数属性如下表所示。

参数属性	必须	描述

参数属性	必须	描述
Type	是	参数的数据类型。取值： • `String`：字符串。例如：`"ecs.s1.medium"`。 • `Number`：整数或浮点数。例如：3.14。 • `CommaDelimitedList`：一组用半角逗号（,）分隔的字符串，可通过Fn::Select函数索引值。例如：`"80, foo, bar"`。 • `Json`：一个JSON格式的字符串。例如：`{"foo": "bar"}`，`[1, 2, 3]`。 • `Boolean`：布尔值。例如：`true`或者`false`。 • `ALIYUN::OOS::Parameter::Value`：存储在OOS参数仓库中的普通参数。例如：`my_image`。 • `ALIYUN::OOS::SecretParameter::Value`：存储在OOS参数仓库中的加密参数。例如：`my_password`。 **说明** `ALIYUN::OOS::Parameter::Value`和`ALIYUN::OOS::SecretParameter::Value`不支持AllowedPattern校验。
Default	否	在创建资源栈时，如果用户没有传入指定值，ROS会检查模板中是否定义默认值。如果已定义默认值，则使用默认值，否则报错。 **说明** 默认值可以设置为`null`，表示该参数取值为空并且忽略对该参数的验证。
AllowedValues	否	包含参数允许值的列表。
AllowedPattern	否	一个正则表达式，用于检查用户输入的字符串类型的参数是否匹配。如果用户输入的不是字符串类型，则报错。 如果使用以下特殊字符，需要在字符前输入两个反斜线（\\）进行转义： `*.?+-$^[ ]( ){ }|\/` **说明** 短划线（-）在紧挨边界时无需转义，例如：[a-z-]。
MaxLength	否	一个整数值，允许String类型使用的字符的最大数目。
MinLength	否	一个整数值，允许String类型使用的字符的最小数目。
MaxValue	否	一个数字值，允许Number类型使用的最大数字值。
MinValue	否	一个数字值，允许Number类型使用的最小数字值。

参数属性	必须	描述
NoEcho	否	当调用查询资源栈时，是否输出参数值。如果将值设置为 `true`，则只输出星号（*）。
Description	否	描述参数的字符串。
ConstraintDescription	否	违反该参数约束条件时，说明该约束条件的字符串。
Label	否	参数别名，支持UTF-8字符。通过模板生成Web表单时，可映射为 `label`。
AssociationProperty	否	用于自动验证参数值的合法性，并且给参数提供可选值。ROS支持的AssociationProperty及其示例，请参见AssociationProperty和AssociationPropertyMetadata。
AssociationPropertyMetadata	否	为AssociationProperty定义约束条件，筛选出符合条件的结果。该属性属于Map类型。AssociationProperty对应的AssociationPropertyMetadata及其示例，请参见AssociationProperty和AssociationPropertyMetadata。
Confirm	否	当NoEcho取值为 `true` 时，参数是否需要二次输入确认。默认值为 `false`。 **说明** 只有String类型的参数，且NoEcho取值为 `true` 时，Confirm可以为 `true`。
TextArea	否	参数是否支持换行。取值： • `true`：支持换行。 • `false`（默认值）：不支持换行。 例如：以下代码表示参数Content支持换行。 `{ "ROSTemplateFormatVersion": "2015-09-01", "Parameters": { "Content": { "Type": "String", "TextArea": true } }, "Outputs": { "TestContent": { "Value": { "Ref": "Content" } } } }`

示例：为Web应用创建资源栈

如果您需要为1个Web应用创建1个资源栈，包含1个负载均衡实例、2个ECS实例和1个RDS实例。如果该Web应用负载较高，可以在创建资源栈时，选择高配的ECS实例；否则可以在创建资源栈时，选择低配的ECS实例。您可以按照如下示例，在模板中定义ECS实例规格参数：

```
"Parameters" : {
  "InstanceType" : {
    "Type" : "String",
    "AllowedValues":["ecs.t1.small","ecs.s1.medium", "ecs.m1.medium", "ecs.c1.large"],
    "Default": "ecs.t1.small",
    "Label": "ECS规格类型",
    "Description" : "请选择创建ECS示例的配置，默认为ecs.t1.small，可选ecs.t1.small, ecs.s1.medium, ecs.m1.medium，ecs.c1.large。"
  }
}
```

示例中，定义的InstanceType参数允许用户在使用模板创建资源栈时，对InstanceType进行重新赋值。如果用户不设置参数值，则使用默认值：`ecs.t1.small`。

在定义资源时，可以引用该参数：

```
"Webserver" : {
  "Type" : "ALIYUN::ECS::Instance",
  "InstanceType": {
    "Ref": "InstanceType"
  }
}
```

6.2.2.2. AssociationProperty和AssociationPropertyMetadata

当您使用ROS创建资源栈管理多种资源时，通常需要打开多个控制台查找资源参数信息。此时您可以在模板的参数配置中指定AssociationProperty以获取所选地域下对应的资源，指定AssociationPropertyMetadata对不同参数添加筛选条件，以便在控制台动态选择参数配置。

参数说明

您可以根据实际需求指定AssociationPropertyMetadata参数的取值：

- 为参数指定特定值。例如：`"RegionId": "cn-hangzhou"`，ROS将填入特定的地域cn-hangzhou。
- 为参数指定变量，格式为`${ParemeterKey}`。例如：`"VpcId": "${VpcId}"`，ROS将动态获取当前模板中参数VpcId对应的值。

> **说明**
>
> - 如果需要将`${ParemeterKey}`指定为固定值，可以增加感叹号（!）。例如：`${!Literal}`表示取值为`${Literal}`。
> - 如果要在Terraform模板中使用参数变量，需要在`$`前增加一个`$`。例如：`"VpcId": "$${VpcId}"`，ROS将动态获取Terraform模板中VpcId对应的值。

ROS支持的AssociationProperty和对应的AssociationPropertyMetadata如下表所示。

AssociationProperty取值	说明	对应的AssociationPropertyMetadata
Password	密码输入。	无
TextArea	富文本输入。	无
Json	对象输入。	无
CommaDelimitedList	逗号分隔的数组。	无
Code	代码输入。	无
FileContent	从本地文件读取内容。	AcceptFileSuffixes：接受上传的文件类型。
Cron	Cron表达式。	无
ChargeType	计费方式。	无
		Overwrite：补充或覆盖内层嵌套参数的定义。 该参数可选，为字典类型，组成部分如下： • 键 内层嵌套参数的路径。路径规则如下： ○ 若要表示list或set中的元素，添加 `*` 。 ○ 若要表示object中的某一项，添加该项的名称。 ○ 路径使用 `.` 分隔。 ○ 开头和中间的 `*` 可以省略。结尾的一个或多个 `*` 不能省略。 示例如下：

AssociationProperty取值	说明	对应的AssociationPropertyMetadata

Auto	ROS会基于Terraform原始数据结构，自动生成AssociationProperty和AssociationPropertyMetadata等字段。
	说明 仅对Terraform类型模板的Json类型参数生效。
		// name的路径为name。
		// address的路径为address。
		variable "user_information" {
		type = object({
		name = string
		address = string
		})
		}
		// region_ids中元素的路径为*。
		variable "region_ids" {
		type = list(string)
		}
		// internal的路径为*.internal，可简写为internal。
		// external的路径为*.external，可简写为external。
		// protocol的路径为*.protocol，可简写为protocol。
		variable "docker_ports" {
		type = list(object({
		internal = number
		external = number
		protocol = string
		}))
		}
		// b的路径为 *.b，可简写为b。
		// b中元素的路径为*.b.*，可简写为b.*
		// c的路径为*.c，可简写为c。
		// d的路径为*.c.*.d，可简写为c.d。
		// d中元素的路径为*.c.*.d.*，可简写为c.d.*。
		// e的路径为*.e，可简写为e。
		// e中元素的路径为*.e.*，可简写为e.*。
		// e中元素的路径为*.e.*.*，可简写为e.*.*（必须保留两个*）。
		// f的路径为*.f，可简写为f。
		// g的路径为*.f.*.*.g，可简写为f.g。
		// g中元素的路径为*.f.*.*.g.*，可简写为f.g.*。
		variable "complex_type_demo" {
		type = list(object({
		b = list(string)
		c = list(object({
		d = list(string)
		}))
		e = list(list(string))
		f = list(list(object({
		g = list(string)
		})))
		}))
		}
		• 值
		参数定义的补充。更多信息，请参见概览。

AssociationProperty取值	说明	对应的AssociationPropertyMetadata

List[Parameter]	参数列表。使用缩进排布。
	仅对Json类型参数生效。
		Parameter：参数定义。
		更多信息，请参见概览。
		示例如下：
		• 示例代码
		<pre>{
		  "ROSTemplateFormatVersion": "2015-09-01",
		  "Parameters": {
		    "RegionIds": {
		      "Label": {
		        "en": "ECS Region Ids",
		        "zh-cn": "ECS地域列表"
		      },
		      "MaxLength": 100,
		      "MinLength": 1,
		      "Type": "Json",
		      "AssociationProperty": "List[Parameter]",
		      "AssociationPropertyMetadata": {
		        "Parameter": {
		          "Type": "String",
		          "AssociationProperty": "ALIYUN::ECS::RegionId",
		          "MinLength": 1,
		          "MaxLength": 64
		        }
		      }
		    }
		  }
		}</pre>• 示例说明
		示例中指定AssociationProperty为List[Parameter]、AssociationPropertyMetadata为Parameter，来实现多个ECS地域ID在控制台缩进排布的效果。

		• Parameters：字典类型，可选，表示一组参数定义。
		  ○ 键：参数名。
		  ○ 值：参数定义。更多信息，请参见概览。
		示例如下：

AssociationProperty取值	说明	对应的AssociationPropertyMetadata
无	使用缩进排布。 仅对Json类型参数生效。	○ 示例代码 `{ "ROSTemplateFormatVersion": "2015-09-01", "Parameters": { "RenameECS": { "Label": { "en": "Rename ECS", "zh-cn": "重命名ECS" }, "MaxLength": 100, "MinLength": 1, "Type": "Json", "AssociationPropertyMetadata": { "Parameters": { "InstanceId": { "Type": "String", "Label": { "en": "ECS Instance Id", "zh-cn": "ECS实例ID" }, "AssociationProperty": "ALIYUN::ECS::Instance::InstanceId", "MinLength": 1, "MaxLength": 64 }, "Name": { "Type": "String", "Label": { "en": "New Name", "zh-cn": "新名称" }, "MinLength": 1 } } } } } }` ○ 示例说明 示例中不指定AssociationProperty，仅指定AssociationPropertyMetadata为Parameters，来实单个ECS实例ID在控制台缩进排布的效果。
		• Metadata：字典类型，可选。更多信息，请参见元数据（Metadata） • Parameters：字典类型，必选，表示一组参数定义。 ○ 键：参数名。 说明 结合List[Parameter]可以实现嵌套结构，且对于嵌套深度没有限制。

AssociationProperty取值	说明	○ 值：参数定义。更多信息，请参见概览。 对应的AssociationPropertyMetadata

List[Parameters]	参数组列表。使用表格排布。
	仅对Json类型参数生效。
		说明 由于使用表格排布，建议使用简单类型参数，不建议进行嵌套。
		
		示例如下：
		○ 示例代码
		
		{
		"ROSTemplateFormatVersion": "2015-09-01",
		"Parameters": {
		"RenameECS": {
		"Label": {
		"en": "Rename ECS",
		"zh-cn": "重命名ECS"
		},
		"MaxLength": 100,
		"MinLength": 1,
		"Type": "Json",
		"AssociationProperty": "List[Parameters]",
		"AssociationPropertyMetadata": {
		"Parameters": {
		"InstanceId": {
		"Type": "String",
		"Label": {
		"en": "ECS Instance Id",
		"zh-cn": "ECS实例ID"
		},
		"AssociationProperty": "ALIYUN::ECS::Instance::InstanceId",
		"MinLength": 1,
		"MaxLength": 64
		},
		"Name": {
		"Type": "String",
		"Label": {
		"en": "New Name",
		"zh-cn": "新名称"
		},
		"MinLength": 1
		}
		}
		}
		}
		}
		}

AssociationProperty取值	说明	对应的AssociationPropertyMetadata。
		○ 示例说明 示例中指定AssociationProperty为List[Parameters]、AssociationPropertyMetadata为Parameters，来实现多个ECS实例ID在控制台表格排布的效果。 • ListMetadata：字典类型，可选，用于控制表格显示。 ○ ShowHeader：bool类型，表示是否显示表头。默认值：true。 ○ ShowRemove：bool类型，表示是否显示移除按钮。默认值：true。
ALIYUN::ECS::RegionId	地域ID。	无○ ShowAddition：bool类型，表示是否新增按钮。默认值：true。 • Order：字符串列表，表示表格从左到右每列对应的参数，默认所有参数按字母序排列显示。 • RegionId：地域ID。 • ShowRandom：展示随机分配选项。
ALIYUN::ECS::ZoneId	可用区ID。	
ALIYUN::ECS::Instance::InstanceId	ECS实例ID。	RegionId：地域ID。
ALIYUN::RDS::Instance::InstanceId	RDS实例ID。	• RegionId：地域ID。 • ZoneId：可用区ID。
ALIYUN::ECS::Disk::DataDiskCategory	数据盘类型。	• RegionId：地域ID。 • ZoneId：可用区ID。 • InstanceType：实例类型。
ALIYUN::ECS::Disk::SystemDiskCategory	系统盘类型。	• RegionId：地域ID。 • ZoneId：可用区ID。 • InstanceType：实例类型。
ALIYUN::ECS::Instance::ImageId	镜像ID。	RegionId：地域ID。
ALIYUN::ECS::VPC::VPCId	专有网络ID。	RegionId：地域ID。
ALIYUN::ECS::VSwitch	交换机。	• RegionId：地域ID。 • ZoneId：可用区ID。 • VpcId：专有网络ID。
ALIYUN::ECS::VSwitch::VSwitchId	交换机ID。	• RegionId：地域ID。 • ZoneId：可用区ID。 • VpcId：专有网络ID。

AssociationProperty取值	说明	对应的AssociationPropertyMetadata
ALIYUN::ECS::Instance::InstanceType	ECS实例规格。	• RegionId：地域ID。 • ZoneId：可用区ID。 • InstanceChargeType：实例付费类型。 • SpotStrategy：按量付费实例的竞价策略。
ALIYUN::ECS::SecurityGroup::SecurityGroupId	安全组ID。	• RegionId：地域ID。 • VpcId：专有网络ID。
ALIYUN::ECS::KeyPair::KeyPairName	密钥对。	RegionId：地域ID。
ALIYUN::ECS::Snapshot::AutoSnapshotPolicyId	自动快照策略。	RegionId：地域ID。
ALIYUN::RDS::Instance::InstanceType	RDS实例规格。	• RegionId：地域ID。 • ZoneId：可用区ID。 • InstanceChargeType：实例付费类型。 • Category：实例系列。 • Engine：数据库类型。 • EngineVersion：数据库版本号。 • DBInstanceClass：实例规格。 • OrderType：订单类型。 • DBInstanceStorageType：实例存储类型。 • DispenseMode：分配模式。
ALIYUN::SLB::Instance::InstanceType	SLB实例规格。	• RegionId：地域ID。 • ZoneId：可用区ID。 • InstanceChargeType：实例付费类型。 • SpotStrategy：按量付费实例的竞价策略。
ALIYUN::SLB::Certificate	SLB证书。	无
ALIYUN::OOS::Template::TemplateName	OOS模板。	RegionId：地域ID。
ALIYUN::OOS::Template::TemplateVersion	OOS模板版本。	• RegionId：地域ID。 • TemplateName：模板名称。
ALIYUN::OOS::Parameter::Value	OOS普通参数。	RegionId：地域ID。
ALIYUN::OOS::SecretParameter::Value	OOS加密参数。	RegionId：地域ID。

AssociationProperty取值	说明	对应的AssociationPropertyMetadata
ALIYUN::OOS::Package::PackageName	OOS软件包。	RegionId：地域ID。
ALIYUN::OOS::Package::PackageVersion	OOS软件包版本。	• RegionId：地域ID。 • TemplateName：模板名称。
ALIYUN::VPC::VirtualBorderRouter::RouteTableId	边界路由器（VBR）。	RegionId：地域ID。
ALIYUN::ESS::ScalingConfiguration::ScalingConfigurationId	弹性伸缩配置。	RegionId：地域ID。
ALIYUN::RAM::User	RAM用户。	RegionId：地域ID。
ALIYUN::RAM::Role	RAM角色。	RegionId：地域ID。

示例1：AssociationProperty示例

在参数中指定AssociationProperty取值为ALIYUN::ECS::Instance::ImageId，以获取所选地域下所有的镜像ID。

```
"Parameters": {
  "UserName": {
    "Label": "用户名",
    "Description": "请输入用户名",
    "Default": "anonymous",
    "Type": "String",
    "MinLength": "6",
    "MaxLength": "12",
    "AllowedValues": [
      "anonymous",
      "user-one",
      "user-two"
    ]
  },
  "PassWord": {
    "Label": "密码",
    "NoEcho": "True",
    "Description": "请输入用户密码",
    "Type": "String",
    "MinLength": "1",
    "MaxLength": "41",
    "AllowedPattern": "[a-zA-Z0-9]*"
  },
  "ImageId": {
    "Label": "镜像",
    "Type": "String",
    "Description": "请选择镜像",
    "AssociationProperty": "ALIYUN::ECS::Instance::ImageId",
    "Default": "centos_7_7_x64_20G_alibase_2020****.vhd"
  }
}
```

参数说明：

- UserName：用户名。String类型，长度为6~12个字符，取值：
  - anonymous（默认值）
  - user-one
  - user-two
- PassWord：密码。String类型，无默认值。长度为1~41个字符，支持大写英文字母、小写英文字母和数字。
  NoEcho取值为true，表示查询资源栈时将不会返回参数值。
- ImageId：镜像ID。String类型。
  AssociationProperty取值为ALIYUN::ECS::Instance::ImageId，创建资源栈时ROS控制台将会验证参数指定的镜像ID是否可用，并以下拉框的方式列出所选地域的其他镜像ID取值。

示例2：AssociationPropertyMetadata示例

在参数中指定AssociationProperty和对应的AssociationPropertyMetadata（RegionId、VpcId和ZoneId），以便获取指定专有网络和可用区的交换机。其中，RegionId取值为固定值cn-hangzhou；VpcId和ZoneId取值为变量${VpcId}和${EcsZone}，将根据ALIYUN::ECS::VPC::VPCId和ALIYUN::ECS::Instance::ZoneId选定的取值进行动态刷新，从而建立交换机与专有网络和可用区的关联关系。

```
{
  "ROSTemplateFormatVersion": "2015-09-01",
  "Parameters": {
    "VpcId": {
      "Type": "String",
      "AssociationProperty": "ALIYUN::ECS::VPC::VPCId"
    },
    "EcsZone": {
      "Type": "String",
      "AssociationProperty": "ALIYUN::ECS::Instance::ZoneId"
    },
    "VSwitchId": {
      "Type": "String",
      "AssociationProperty": "ALIYUN::ECS::VSwitch::VSwitchId",
      "AssociationPropertyMetadata": {
           "RegionId": "cn-hangzhou",
           "VpcId": "${VpcId}",
           "ZoneId": "${EcsZone}"
      }
    }
  }
}
```

示例3：Terraform自动转换示例

当AssociationProperty取值为Auto时，ROS会基于Terraform原始数据结构，自动生成AssociationProperty和AssociationPropertyMetadata等字段。

- 完整模板示例代码

```
variable "user_information" {
  type = object({
    name    = string
    address = string
  })
  description = <<EOT
  {
    "AssociationProperty": "Auto",
    "AssociationPropertyMetadata": {
      "Overwrite": {
        "name": {
          "Label": {
            "zh-cn": "姓名",
            "en": "Name"
          }
        }
      }
    }
  }
  EOT
}
variable "region_ids" {
  type    = list(string)
  description = <<EOT
  [
```

```
  {
    "AssociationProperty": "Auto",
    "AssociationPropertyMetadata": {
      "Overwrite": {
        "*": {
          "AssociationProperty": "ALIYUN::ECS::RegionId"
        }
      }
    }
  }
  EOT
}
variable "docker_ports" {
  type = list(object({
    internal = number
    external = number
    protocol = string
  }))
  description = <<EOT
  {
    "AssociationProperty": "Auto",
    "AssociationPropertyMetadata": {
      "Overwrite": {
        "protocol": {
          "AllowedValues": ["TCP", "UDP", "ICMP"]
        },
        "internal": {
          "Label": {
            "zh-cn": "内部端口",
            "en": "Internal Port"
          }
        }
      }
    }
  }
  EOT
}
variable "complex_type_demo" {
  type = list(object({
    b = list(string)
    c = list(object({
      d = list(string)
    }))
    e = list(list(string))
    f = list(list(object({
      g = list(string)
    })))
  }))
  description = <<EOT
  {
    "AssociationProperty": "Auto",
    "AssociationPropertyMetadata": {
        "Overwrite": {
            "b": {
                "MaxLength": 3,
```

```
                "AssociationPropertyMetadata": {
                    "Something": "123"
                },
                "Label": {
                    "zh-cn": "B",
                    "en": "B"
                }
            },
            "b.*": {
                "AssociationProperty": "ALIYUN::ECS::RegionId"
            },
            "c": {
                "Label": {
                    "zh-cn": "C",
                    "en": "C"
                }
            },
            "c.d": {
                "Label": {
                    "zh-cn": "C/D",
                    "en": "C/D"
                }
            },
            "c.d.*": {
                "AssociationProperty": "ALIYUN::ECS::RegionId"
            },
            "e": {
                "Label": {
                    "zh-cn": "E",
                    "en": "E"
                }
            },
            "e.*": {
                "MaxLength": 3
            },
            "e.*.*": {
                "AssociationProperty": "ALIYUN::ECS::RegionId"
            },
            "f.g": {
                "Label": {
                    "zh-cn": "F/G",
                    "en": "F/G"
                }
            },
            "f.g.*": {
                "AssociationProperty": "ALIYUN::ECS::RegionId"
            }
        }
    }
  }
  EOT
}
```

- 参数user_information转换后模板示例代码

```
{
  "AssociationPropertyMetadata":{
    "Parameters":{
      "address":{
        "Type":"String"
      },
      "name":{
        "Type":"String",
        "Label":{
          "en":"Name",
          "zh-cn":"姓名"
        }
      }
    }
  }
}
```

- 参数region_ids转换后模板示例代码

```
{
  "AssociationPropertyMetadata":{
    "Parameter":{
      "AssociationProperty":"ALIYUN::ECS::RegionId",
      "Type":"String"
    }
  },
  "AssociationProperty":"List[Parameter]"
}
```

- 参数docker_ports转换后模板示例代码

```
{
  "AssociationPropertyMetadata":{
    "Parameters":{
      "internal":{
        "Type":"Number",
        "Label":{
          "en":"Internal Port",
          "zh-cn":"内部端口"
        }
      },
      "protocol":{
        "Type":"String",
        "AllowedValues":[
          "TCP",
          "UDP",
          "ICMP"
        ]
      },
      "external":{
        "Type":"Number"
      }
    }
  },
  "AssociationProperty":"List[Parameters]"
}
```

- 参数complex_type_demo转换后模板示例代码

```
{
  "AssociationPropertyMetadata":{
    "Parameter":{
      "AssociationPropertyMetadata":{
        "Parameters":{
          "b":{
            "AssociationPropertyMetadata":{
              "Parameter":{
                "AssociationProperty":"ALIYUN::ECS::RegionId",
                "Type":"String"
              },
              "Something":"123"
            },
            "AssociationProperty":"List[Parameter]",
            "Type":"Json",
            "Label":{
              "en":"B",
              "zh-cn":"B"
            },
            "MaxLength":3
          },
          "c":{
            "AssociationPropertyMetadata":{
              "Parameter":{
                "AssociationPropertyMetadata":{
                  "Parameters":{
```

```
                "d":{
                  "AssociationPropertyMetadata":{
                    "Parameter":{
                      "AssociationProperty":"ALIYUN::ECS::RegionId",
                      "Type":"String"
                    }
                  },
                  "AssociationProperty":"List[Parameter]",
                  "Type":"Json",
                  "Label":{
                    "en":"C/D",
                    "zh-cn":"C/D"
                  }
                }
              }
            },
            "Type":"Json"
          }
        },
        "AssociationProperty":"List[Parameter]",
        "Type":"Json",
        "Label":{
          "en":"C",
          "zh-cn":"C"
        }
      },
      "e":{
        "AssociationPropertyMetadata":{
          "Parameter":{
            "AssociationPropertyMetadata":{
              "Parameter":{
                "AssociationProperty":"ALIYUN::ECS::RegionId",
                "Type":"String"
              }
            },
            "AssociationProperty":"List[Parameter]",
            "Type":"Json",
            "MaxLength":3
          }
        },
        "AssociationProperty":"List[Parameter]",
        "Type":"Json",
        "Label":{
          "en":"E",
          "zh-cn":"E"
        }
      },
      "f":{
        "AssociationPropertyMetadata":{
          "Parameter":{
            "AssociationPropertyMetadata":{
              "Parameter":{
                "AssociationPropertyMetadata":{
                  "Parameters":{
                    "~":{
```

```
                        "g":{
                          "AssociationPropertyMetadata":{
                            "Parameter":{
                              "AssociationProperty":"ALIYUN::ECS::RegionId",
                              "Type":"String"
                            }
                          },
                          "AssociationProperty":"List[Parameter]",
                          "Type":"Json",
                          "Label":{
                            "en":"F/G",
                            "zh-cn":"F/G"
                          }
                        }
                      }
                    },
                    "Type":"Json"
                  }
                },
                "AssociationProperty":"List[Parameter]",
                "Type":"Json"
              }
            },
            "AssociationProperty":"List[Parameter]",
            "Type":"Json"
          }
        },
        "Type":"Json"
      }
    },
    "AssociationProperty":"List[Parameter]"
}
```

更多示例

您也可以使用AssociationProperty和AssociationPropertyMetadata，实现以下诉求：

- 基于Mappings自动化配置CentOS系统的yum源
- 在资源编排控制台动态选择参数配置
- 基于参数关联关系动态呈现参数

6.2.2.3. 参数取值本地化

资源编排服务ROS（Resource Orchestration Service）支持对参数取值进行本地化，根据控制台语言环境将参数取值翻译为对应的语言。目前支持本地化的语言有中文和英文。

模板语法

您可以参考以下格式定义参数，在 `LocaleKey` 中定义本地化键索引，在 `AllowedValues` 中定义参数取值列表，从而在控制台实现根据语言呈现取值列表的显示效果。

```
{
  "ROSTemplateFormatVersion": "2015-09-01",
  "Parameters": {
    "ECSInternetChargeType": {
      "Type": "String",
      "AllowedValues": ["PayByBandwidth", "PayByTraffic"],
      "AssociationPropertyMetadata": {
        "LocaleKey": "InternetChargeType"
      }
    }
  }
}
```

中文控制台显示效果：

英文控制台显示效果：

参数取值列表

下表为您列出参数的取值列表和示例，参数值的中文含义以ROS中文控制台实际显示为准。

参数	本地化键索引	参数值	示例
包年包月周期	PricingCycle	• Week • week • Month • month • Year • year	`{ "ECSPeriodUnit": { "Type": "String", "AllowedValues": ["Week", "Month", "Year"], "AssociationPropertyMetadata": { "LocaleKey": "PricingCycle" } } }`

参数	本地化键索引	参数值	示例

付费模式	InstanceChargeType	• PostPaid
		• Postpaid
		• PayOnDemand
		• PayAsYouGo
		• POSTPAY
		• PrePaid
		• Prepaid
		• PrePay
		• Subscription
		• PREPAY
			<pre>{
			  "ECSInstanceChargeType": {
			    "Type": "String",
			    "AllowedValues": ["PostPaid", "PrePaid"],
			    "AssociationPropertyMetadata": {
			        "LocaleKey": "InstanceChargeType"
			      }
			  }
			}</pre>

带宽计费方式	InternetChargeType	• PayByBandwidth
		• paybybandwidth
		• PayBySpec
		• PayByTraffic
		• paybytraffic
		• PayByLcu
			<pre>{
			  "ECSInternetChargeType": {
			    "Type": "String",
			    "AllowedValues": ["PayByBandwidth", "PayByTraffic"],
			    "AssociationPropertyMetadata": {
			        "LocaleKey": "InternetChargeType"
			      }
			  }
			}</pre>

网络类型	NetType	• Intranet
		• Internet
		• privatenet
		• pubnet
			<pre>{
			  "ECSInstanceNetType": {
			    "Type": "String",
			    "AllowedValues": ["Intranet", "Internet"],
			    "AssociationPropertyMetadata": {
			        "LocaleKey": "NetType"
			      }
			  }
			}</pre>

参数	本地化键索引	参数值	示例

NAT网关的规格	NatGatewaySpec	• Small
		• Middle
		• Large
		• XLarge.1
			<pre>{
			  "NatGatewaySpec": {
			    "Type": "String",
			    "AllowedValues": ["Small", "Middle", "Large", "XLarge.1"],
			    "AssociationPropertyMetadata": {
			        "LocaleKey": "NatGatewaySpec"
			      }
			  }
			}</pre>

布尔类型的参数	ParameterBoolean	• True
		• true
		• False
		• false
		• on
		• off
		• ConsoleProtection
		• NonProtection
		• Enable
		• Disabled
		• Enabled
			<pre>{
			  "ECSAutoRenew": {
			    "Type": "String",
			    "AllowedValues": ["true", "false"],
			    "AssociationPropertyMetadata": {
			        "LocaleKey": "ParameterBoolean"
			      }
			  }
			}</pre>

实例的竞价策略	InstanceSpotStrategy	• Spot
		• NoSpot
		• SpotWithPriceLimit
		• SpotAsPriceGo
			<pre>{
			  "ECSSpotStrategy": {
			    "Type": "String",
			    "AllowedValues": ["Spot", "NoSpot"],
			    "AssociationPropertyMetadata": {
			        "LocaleKey": "InstanceSpotStrategy"
			      }
			  }
			}</pre>

参数	本地化键索引	参数值	示例

实例的网络类型	InstanceNetworkType	• vpc
		• VPC
		• classic
		• CLASSIC
		• Classic
			<pre>{
			  "RDSNetworkType": {
			    "Type": "String",
			    "AllowedValues": ["VPC", "CLASSIC"],
			    "AssociationPropertyMetadata": {
			        "LocaleKey":
			"InstanceNetworkType"
			      }
			  }
			}</pre>

ECS是否启用安全加固	ECSSecurityEnhancementStrategy	• Active
		• Deactive
			<pre>{
			  "ECSSecurityEnhancementStrategy": {
			    "Type": "String",
			    "AllowedValues": ["Active",
			"Deactive"],
			    "AssociationPropertyMetadata": {
			        "LocaleKey":
			"ECSSecurityEnhancementStrategy"
			      }
			  }
			}</pre>

ECS抢占式实例的策略	ECSTargetCapacity	• lowest-price
		• diversified
			<pre>{
			  "ECSTargetCapacity": {
			    "Type": "String",
			    "AllowedValues": ["lowest-price",
			"diversified"],
			    "AssociationPropertyMetadata": {
			        "LocaleKey": "ECSTargetCapacity"
			      }
			  }
			}</pre>

参数	本地化键索引	参数值	示例

ECS停止超额抢占式实例后的下一步动作	ECSSpotInstanceInterruptionBehavior	• stop
		• terminate
			<pre>{
			  "ECSSpotInstanceInterruptionBehavior": {
			    "Type": "String",
			    "AllowedValues": ["stop", "terminate"],
			    "AssociationPropertyMetadata": {
			        "LocaleKey": "ECSSpotInstanceInterruptionBehavior"
			    }
			  }
			}</pre>

ECS命令的类型	ECSCommandType	• RunBatScript
		• RunShellScript
			<pre>{
			  "ECSCommandType": {
			    "Type": "String",
			    "AllowedValues": ["RunBatScript", "RunPowerShellScript", "RunShellScript"],
			    "AssociationPropertyMetadata": {
			        "LocaleKey": "ECSCommandType"
			    }
			  }
			}</pre>

ECS弹性供应组的交付类型	ECSAutoProvisioningGroupType	• request
		• maintain
			<pre>{
			  "ECSAutoProvisioningGroupType": {
			    "Type": "String",
			    "AllowedValues": ["request", "maintain"],
			    "AssociationPropertyMetadata": {
			        "LocaleKey": "ECSAutoProvisioningGroupType"
			    }
			  }
			}</pre>

参数	本地化键索引	参数值	示例

ECS部署集内实例宕机迁移后，缺乏可供打散的实例库存的紧急处理方案	ECSOnUnableToRedeployFailedInstance	• CancelMembershipAndStart
		• KeepStopped
			<pre>{
			  "ECSOnUnableToRedeployFailedInstance": {
			    "Type": "String",
			    "AllowedValues": ["CancelMembershipAndStart", "KeepStopped"],
			    "AssociationPropertyMetadata": {
			        "LocaleKey": "ECSOnUnableToRedeployFailedInstance"
			      }
			  }
			}</pre>

NAS存储类型	NasStorageType	• Performance
		• Capacity
		• standard
		• advance
			<pre>{
			  "NasStorageType": {
			    "Type": "String",
			    "AllowedValues": ["Performance", "Capacity"],
			    "AssociationPropertyMetadata": {
			        "LocaleKey": "NasStorageType"
			      }
			  }
			}</pre>

NAS文件系统类型	NasFileSystemType	• standard
		• extreme
		• cpfs
			<pre>{
			  "NasFileSystemType": {
			    "Type": "String",
			    "AllowedValues": ["standard", "extreme", "cpfs"],
			    "AssociationPropertyMetadata": {
			        "LocaleKey": "NasFileSystemType"
			      }
			  }
			}</pre>

参数	本地化键索引	参数值	示例

云盘的性能等级	DiskPerformanceLevel	• PL0
		• PL1
		• PL2
		• PL3
			<pre>{
			  "DiskPerformanceLevel": {
			    "Type": "String",
			    "AllowedValues": ["PL0", "PL1", "PL2", "PL3"],
			    "AssociationPropertyMetadata": {
			        "LocaleKey": "DiskPerformanceLevel"
			      }
			  }
			}</pre>

SLB克隆策略	SlbBackendServersPolicy	• clone
		• empty
		• append
		• replace
			<pre>{
			  "SlbBackendServersPolicy": {
			    "Type": "String",
			    "AllowedValues": ["clone", "empty", "append", "replace"],
			    "AssociationPropertyMetadata": {
			        "LocaleKey": "SlbBackendServersPolicy"
			      }
			  }
			}</pre>

SLB后端服务器类型	SlbBackendServersType	• ecs
		• eni
		• eci
			<pre>{
			  "SlbBackendServersType": {
			    "Type": "String",
			    "AllowedValues": ["ecs", "eni", "eci"],
			    "AssociationPropertyMetadata": {
			        "LocaleKey": "SlbBackendServersType"
			      }
			  }
			}</pre>

参数	本地化键索引	参数值	示例

EIP线路类型	EIPIsp	• BGP
		• BGP_PRO
		• ChinaTelecom
		• ChinaUnicom
		• ChinaMobile
		• BGP_FinanceCloud
			<pre>{
			  "EIPIsp": {
			    "Type": "String",
			    "AllowedValues": ["BGP", "BGP_PRO", "ChinaTelecom", "ChinaUnicom", "ChinaMobile", "BGP_FinanceCloud"],
			    "AssociationPropertyMetadata": {
			        "LocaleKey": "EIPIsp"
			      }
			  }
			}</pre>

EIP绑定模式	EIPAssociationMode	• NAT
		• MULTI_BINDED
		• BINDED
			<pre>{
			  "EIPAssociationMode": {
			    "Type": "String",
			    "AllowedValues": ["NAT", "MULTI_BINDED", "BINDED"],
			    "AssociationPropertyMetadata": {
			        "LocaleKey": "EIPAssociationMode"
			      }
			  }
			}</pre>

EIPSegment网络类型	EIPSegmentNetmode	• public
		• hybrid
			<pre>{
			  "EIPSegmentNetmode": {
			    "Type": "String",
			    "AllowedValues": ["public", "hybrid"],
			    "AssociationPropertyMetadata": {
			        "LocaleKey": "EIPSegmentNetmode"
			      }
			  }
			}</pre>

参数	本地化键索引	参数值	示例

FlowLog要捕获流量的资源类型	FlowLogResourceType	• NetworkInterface
		• VSwitch
		• VPC
			<pre>{
			  "EIPAssociationMode": {
			    "Type": "String",
			    "AllowedValues": ["NetworkInterface", "VSwitch", "VPC"],
			    "AssociationPropertyMetadata": {
			        "LocaleKey": "EIPAssociationMode"
			      }
			  }
			}</pre>

FlowLog采集的流量类型	FlowLogTrafficType	• All
		• Allow
		• Drop
			<pre>{
			  "FlowLogTrafficType": {
			    "Type": "String",
			    "AllowedValues": ["All", "Allow", "Drop"],
			    "AssociationPropertyMetadata": {
			        "LocaleKey": "FlowLogTrafficType"
			      }
			  }
			}</pre>

GrantInstanceToCen网络实例类型	GrantInstanceToCenInstanceType	• VPC
		• VBR
		• CCN
			<pre>{
			  "GrantInstanceToCenInstanceType": {
			    "Type": "String",
			    "AllowedValues": ["VPC", "VBR", "CCN"],
			    "AssociationPropertyMetadata": {
			        "LocaleKey": "GrantInstanceToCenInstanceType"
			      }
			  }
			}</pre>

参数	本地化键索引	参数值	示例

NAT网关的类型	NatType	• Normal
		• Enhanced
			<pre>{
			  "NatType": {
			    "Type": "String",
			    "AllowedValues": ["Normal",
			"Enhanced"],
			    "AssociationPropertyMetadata": {
			        "LocaleKey": "NatType"
			      }
			  }
			}</pre>

RouterInterface所属的路由器类型	RouterType	• VRouter
		• VBR
			<pre>{
			  "RouterType": {
			    "Type": "String",
			    "AllowedValues": ["VRouter", "VBR"],
			    "AssociationPropertyMetadata": {
			        "LocaleKey": "RouterType"
			      }
			  }
			}</pre>

RDS账号类型	RDSAccountType	• Normal
		• Super
		• Sysadmin
			<pre>{
			  "RDSAccountType": {
			    "Type": "String",
			    "AllowedValues": ["Normal", "Super",
			"Sysadmin"],
			    "AssociationPropertyMetadata": {
			        "LocaleKey": "RDSAccountType"
			      }
			  }
			}</pre>

参数	本地化键索引	参数值	示例

RDS账号权限	RDSAccountPrivilege	• ReadWrite
		• ReadOnly
		• DDLOnly
		• DMLOnly
		• DBOwner
			<pre>{
			  "RDSAccountPrivilege": {
			    "Type": "String",
			    "AllowedValues": ["ReadWrite", "ReadOnly", "DDLOnly", "DMLOnly"],
			    "AssociationPropertyMetadata": {
			        "LocaleKey": "RDSAccountPrivilege"
			      }
			  }
			}</pre>

RDS实例连接地址的类型	RDSConnectionStringType	• Inner
		• Public
			<pre>{
			  "RDSConnectionStringType": {
			    "Type": "String",
			    "AllowedValues": ["Inner", "Public"],
			    "AssociationPropertyMetadata": {
			        "LocaleKey": "RDSConnectionStringType"
			      }
			  }
			}</pre>

RDS实例系列	RDSCategory	• Basic
		• HighAvailability
		• AlwaysOn
		• Finance
			<pre>{
			  "RDSCategory": {
			    "Type": "String",
			    "AllowedValues": ["Basic", "HighAvailability", "AlwaysOn", "Finance"],
			    "AssociationPropertyMetadata": {
			        "LocaleKey": "RDSCategory"
			      }
			  }
			}</pre>

参数	本地化键索引	参数值	示例

RDS数据库的连接模式	RDSConnectionMode	• Standard
		• Safe
			<pre>{
			  "RDSConnectionMode": {
			    "Type": "String",
			    "AllowedValues": ["Standard", "Safe"],
			    "AssociationPropertyMetadata": {
			        "LocaleKey": "RDSConnectionMode"
			      }
			  }
			}</pre>

RDS数据库备份类型	RDSBackupPolicyMode	• DataBackupPolicy
		• LogBackupPolicy
		• FullBackup
		• IncrementalBackup
			<pre>{
			  "RDSBackupPolicyMode": {
			    "Type": "String",
			    "AllowedValues": ["FullBackup", "IncrementalBackup"],
			    "AssociationPropertyMetadata": {
			        "LocaleKey": "RDSBackupPolicyMode"
			      }
			  }
			}</pre>

RDS数据是否启用二级备份功能	RDSBackUpCategory	• Flash
		• Standard
			<pre>{
			  "RDSBackUpCategory": {
			    "Type": "String",
			    "AllowedValues": ["Flash", "Standard"],
			    "AssociationPropertyMetadata": {
			        "LocaleKey": "RDSBackUpCategory"
			      }
			  }
			}</pre>

参数	本地化键索引	参数值	示例

RDS数据库释放后的归档备份保留策略	RDSReleasedKeepPolicy	• Lastest
		• All
			<pre>{
			  "RDSReleasedKeepPolicy": {
			    "Type": "String",
			    "AllowedValues": ["Lastest", "All"],
			    "AssociationPropertyMetadata": {
			        "LocaleKey":
			"RDSReleasedKeepPolicy"
			      }
			  }
			}</pre>

RDS实例的安全套接层（SSL）链接设置	RDSSSLSetting	• Disabled
		• EnabledForPublicConnection
		• EnabledForInnerConnection
			<pre>{
			  "RDSSSLSetting": {
			    "Type": "String",
			    "AllowedValues": ["Disabled",
			"EnabledForPublicConnection",
			"EnabledForInnerConnection"],
			    "AssociationPropertyMetadata": {
			        "LocaleKey": "RDSSSLSetting"
			      }
			  }
			}</pre>

Redis账号权限	RedisAccountPrivilege	• RoleReadOnly
		• RoleReadWrite
		• RoleRepl
			<pre>{
			  "RedisAccountPrivilege": {
			    "Type": "String",
			    "AllowedValues": ["RoleReadOnly",
			"RoleReadWrite", "RoleRepl"],
			    "AssociationPropertyMetadata": {
			        "LocaleKey":
			"RedisAccountPrivilege"
			      }
			  }
			}</pre>

参数	本地化键索引	参数值	示例

Redis数据逐出策略	RedisEvictionPolicy	• noeviction
		• allkeys-lru
		• volatile-lru
		• allkeys-random
		• volatile-random
		• volatile-ttl
			<pre>{
			  "RedisEvictionPolicy": {
			    "Type": "String",
			    "AllowedValues": ["noeviction", "allkeys-lru", "volatile-lru", "allkeys-random", "volatile-random", "volatile-ttl"],
			    "AssociationPropertyMetadata": {
			        "LocaleKey": "RedisEvictionPolicy"
			      }
			  }
			}</pre>

Redis SSL状态	RedisSSLEnabled	• Disable
		• Enable
		• Update
			<pre>{
			  "RedisSSLEnabled": {
			    "Type": "String",
			    "AllowedValues": ["Disable", "Enable", "Update"],
			    "AssociationPropertyMetadata": {
			        "LocaleKey": "RedisSSLEnabled"
			      }
			  }
			}</pre>

OSS访问权限	OSSAccessControl	• private
		• public-read
		• public-read-write
			<pre>{
			  "OSSAccessControl": {
			    "Type": "String",
			    "AllowedValues": ["private", "public-read", "public-read-write"],
			    "AssociationPropertyMetadata": {
			        "LocaleKey": "OSSAccessControl"
			      }
			  }
			}</pre>

参数	本地化键索引	参数值	示例

OSS存储空间类型	OSSStorageClass	• Standard
		• IA
		• Archive
			<pre>{
			  "OSSStorageClass": {
			    "Type": "String",
			    "AllowedValues": ["Standard", "IA", "Archive"],
			    "AssociationPropertyMetadata": {
			        "LocaleKey": "OSSStorageClass"
			    }
			  }
			}</pre>

OOS参数的约束条件	OOSConstraints	• AllowedValues
		• AllowedPattern
		• MinLength
		• MaxLength
			<pre>{
			  "OOSConstraints": {
			    "Type": "String",
			    "AllowedValues": ["MinLength", "MaxLength"],
			    "AssociationPropertyMetadata": {
			        "LocaleKey": "OOSConstraints"
			    }
			  }
			}</pre>

MongoDB存储引擎	MongoDBStorageEngine	• WiredTiger
		• RocksDB
		• TerarkDB
			<pre>{
			  "MongoDBStorageEngine": {
			    "Type": "String",
			    "AllowedValues": ["WiredTiger", "RocksDB", "TerarkDB"],
			    "AssociationPropertyMetadata": {
			        "LocaleKey": "MongoDBStorageEngine"
			    }
			  }
			}</pre>

参数	本地化键索引	参数值	示例

CEN网络实例的类型	CenChildInstanceType	• VPC
		• VBR
		• CCN
			<pre>{
			  "CenChildInstanceType": {
			    "Type": "String",
			    "AllowedValues": ["VPC", "VBR", "CCN"],
			    "AssociationPropertyMetadata": {
			        "LocaleKey": "CenChildInstanceType"
			      }
			  }
			}</pre>

CEN网络实例所属的地域	CenGeographicRegionId	• China
		• North-America
		• Asia-Pacific
		• Europe
		• Australia
			<pre>{
			  "CenGeographicRegionId": {
			    "Type": "String",
			    "AllowedValues": ["China", "North-America", "Asia-Pacific", "Europe", "Australia"],
			    "AssociationPropertyMetadata": {
			        "LocaleKey": "CenGeographicRegionId"
			      }
			  }
			}</pre>

参数	本地化键索引	参数值	示例

CEN路由策略应用的方向	CenTransmitDirection	• RegionIn
		• RegionOut
			<pre>{
			  "CenTransmitDirection": {
			    "Type": "String",
			    "AllowedValues": ["RegionIn", "RegionOut"],
			    "AssociationPropertyMetadata": {
			        "LocaleKey": "CenTransmitDirection"
			      }
			  }
			}</pre>

CEN匹配模式	CenMatchMode	• Include
		• Complete
			<pre>{
			  "CenMatchMode": {
			    "Type": "String",
			    "AllowedValues": ["Include", "Complete"],
			    "AssociationPropertyMetadata": {
			        "LocaleKey": "CenMatchMode"
			      }
			  }
			}</pre>

CEN所有匹配条件通过后的策略行为	CenMapResult	• Permit
		• Deny
			<pre>{
			  "CenMapResult": {
			    "Type": "String",
			    "AllowedValues": ["Permit", "Deny"],
			    "AssociationPropertyMetadata": {
			        "LocaleKey": "CenMapResult"
			      }
			  }
			}</pre>

CEN匹配路由的类型列表	CenRouteTypes	• System
		• Custom
		• BGP
			<pre>{
			  "CenRouteTypes": {
			    "Type": "String",
			    "AllowedValues": ["System", "Custom", "BGP"],
			    "AssociationPropertyMetadata": {
			        "LocaleKey": "CenRouteTypes"
			      }
			  }
			}</pre>

参数	本地化键索引	参数值	示例

CEN操作Community的模式	CenCommunityOperateMode	• Additive
		• Replace
			<pre>{
			  "CenCommunityOperateMode": {
			    "Type": "String",
			    "AllowedValues": ["Additive", "Replace"],
			    "AssociationPropertyMetadata": {
			        "LocaleKey": "CenCommunityOperateMode"
			      }
			  }
			}</pre>

CEN路由条目的下一跳类型	CenNextHopType	• BlackHole
		• Attachment
			<pre>{
			  "CenNextHopType": {
			    "Type": "String",
			    "AllowedValues": ["BlackHole", "Attachment"],
			    "AssociationPropertyMetadata": {
			        "LocaleKey": "CenNextHopType"
			      }
			  }
			}</pre>

CS节点CPU管理策略	CSCpuPolicy	• static
		• none
			<pre>{
			  "CSCpuPolicy": {
			    "Type": "String",
			    "AllowedValues": ["static", "none"],
			    "AssociationPropertyMetadata": {
			        "LocaleKey": "CSCpuPolicy"
			      }
			  }
			}</pre>

CS调度策略	CSTaintsEffect	• NoSchedule
		• NoExecute
		• PreferNoSchedule
			<pre>{
			  "CSTaintsEffect": {
			    "Type": "String",
			    "AllowedValues": ["NoSchedule", "NoExecute", "PreferNoSchedule"],
			    "AssociationPropertyMetadata": {
			        "LocaleKey": "CSTaintsEffect"
			      }
			  }
			}</pre>

参数	本地化键索引	参数值	示例

自动伸缩类型	AutoScalingType	• cpu
		• gpu
		• gpushare
		• spot
			<pre>{
			  "AutoScalingType": {
			    "Type": "String",
			    "AllowedValues": ["cpu", "gpu", "gpushare", "spot"],
			    "AssociationPropertyMetadata": {
			        "LocaleKey": "AutoScalingType"
			      }
			  }
			}</pre>

伸缩组模式	ScalingPolicy	• release
		• recycle
			<pre>{
			  "ScalingPolicy": {
			    "Type": "String",
			    "AllowedValues": ["release", "recycle"],
			    "AssociationPropertyMetadata": {
			        "LocaleKey": "ScalingPolicy"
			      }
			  }
			}</pre>

ESS报警类型	ESSAlarmTaskMetricType	• system
		• custom
			<pre>{
			  "ESSAlarmTaskMetricType": {
			    "Type": "String",
			    "AllowedValues": ["system", "custom"],
			    "AssociationPropertyMetadata": {
			        "LocaleKey": "ESSAlarmTaskMetricType"
			      }
			  }
			}</pre>

参数	本地化键索引	参数值	示例

ESS报警统计方法	ESSAlarmTaskMetricType	• Average
		• Minimum
		• Maximum
			<pre>{
			  "ESSAlarmTaskMetricType": {
			    "Type": "String",
			    "AllowedValues": ["Average",
			"Minimum", "Maximum"],
			    "AssociationPropertyMetadata": {
			        "LocaleKey":
			"ESSAlarmTaskMetricType"
			      }
			  }
			}</pre>

ESS生命周期挂钩	ESSLifecycleHookDefaultResult	• CONTINUE
		• ABANDON
			<pre>{
			  "ESSLifecycleHookDefaultResult": {
			    "Type": "String",
			    "AllowedValues": ["CONTINUE",
			"ABANDON"],
			    "AssociationPropertyMetadata": {
			        "LocaleKey":
			"ESSLifecycleHookDefaultResult"
			      }
			  }
			}</pre>

ESS生命周期挂钩适用的伸缩活动类型	ESSLifecycleHookLifecycleTransition	• SCALE_OUT
		• SCALE_IN
			<pre>{
			  "ESSLifecycleHookLifecycleTransition":
			{
			    "Type": "String",
			    "AllowedValues": ["SCALE_OUT",
			"SCALE_IN"],
			    "AssociationPropertyMetadata": {
			        "LocaleKey":
			"ESSLifecycleHookLifecycleTransition"
			      }
			  }
			}</pre>

参数	本地化键索引	参数值	示例

ESS伸缩组弹性收缩活动	ESSRemovalPolicys	• OldestInstance
		• NewestInstance
		• OldestScalingConfiguration
			<pre>{
			  "ESSRemovalPolicys": {
			    "Type": "String",
			    "AllowedValues": ["OldestInstance", "NewestInstance", "OldestScalingConfiguration"],
			    "AssociationPropertyMetadata": {
			        "LocaleKey": "ESSRemovalPolicys"
			      }
			  }
			}</pre>

ESS多可用区伸缩组ECS实例扩缩容策略	ESSMultiAZPolicy	• PRIORITY
		• BALANCE
		• COST_OPTIMIZED
			<pre>{
			  "ESSMultiAZPolicy": {
			    "Type": "String",
			    "AllowedValues": ["PRIORITY", "BALANCE", "COST_OPTIMIZED"],
			    "AssociationPropertyMetadata": {
			        "LocaleKey": "ESSMultiAZPolicy"
			      }
			  }
			}</pre>

ESS伸缩规则的调整方式	ESSAdjustmentType	• QuantityChangeInCapacity
		• PercentChangeInCapacity
		• TotalCapacity
			<pre>{
			  "ESSAdjustmentType": {
			    "Type": "String",
			    "AllowedValues": ["QuantityChangeInCapacity", "PercentChangeInCapacity", "TotalCapacity"],
			    "AssociationPropertyMetadata": {
			        "LocaleKey": "ESSAdjustmentType"
			      }
			  }
			}</pre>

参数	本地化键索引	参数值	示例

ESS伸缩规则的预定义监控项	ESSMetricName	• CpuUtilization
		• ClassicInternetRx
		• ClassicInternetTx
		• VpcInternetRx
		• VpcInternetTx
		• IntranetRx
		• IntranetTx
			<pre>{
			  "ESSMetricName": {
			    "Type": "String",
			    "AllowedValues": ["CpuUtilization",
			"ClassicInternetRx",
			"ClassicInternetTx", "VpcInternetRx",
			"VpcInternetTx", "IntranetRx",
			"IntranetTx"],
			    "AssociationPropertyMetadata": {
			        "LocaleKey": "ESSMetricName"
			      }
			  }
			}</pre>

ESS伸缩规则类型	ESSScalingRuleType	• SimpleScalingRule
		• TargetTrackingScalingRule
		• StepScalingRule
		• PredictiveScalingRule
			<pre>{
			  "ESSScalingRuleType": {
			    "Type": "String",
			    "AllowedValues":
			["SimpleScalingRule",
			"TargetTrackingScalingRule",
			"StepScalingRule",
			"PredictiveScalingRule"],
			    "AssociationPropertyMetadata": {
			        "LocaleKey":
			"ESSScalingRuleType"
			      }
			  }
			}</pre>

参数	本地化键索引	参数值	示例

ESS预测规则最大值处理方式	ESSPredictiveValueBehavior	• MaxOverridePredictiveValue
		• PredictiveValueOverrideMax
		• PredictiveValueOverrideMaxWithBuffer
			<pre>{
			  "ESSPredictiveValueBehavior": {
			    "Type": "String",
			    "AllowedValues":
			["MaxOverridePredictiveValue",
			"PredictiveValueOverrideMax",
			"PredictiveValueOverrideMaxWithBuffer"],
			    "AssociationPropertyMetadata": {
			        "LocaleKey":
			"ESSPredictiveValueBehavior"
			      }
			  }
			}</pre>

ESS预测规则的模式	ESSPredictiveScalingMode	• PredictAndScale
		• PredictOnly
			<pre>{
			  "ESSPredictiveScalingMode": {
			    "Type": "String",
			    "AllowedValues": ["PredictAndScale",
			"PredictOnly"],
			    "AssociationPropertyMetadata": {
			        "LocaleKey":
			"ESSPredictiveScalingMode"
			      }
			  }
			}</pre>

ESS定时任务	ESSScheduledTaskRecurrenceType	• Daily
		• Weekly
		• Monthly
		• Cron
			<pre>{
			  "ESSScheduledTaskRecurrenceType": {
			    "Type": "String",
			    "AllowedValues": ["Daily", "Weekly",
			"Monthly", "Cron"],
			    "AssociationPropertyMetadata": {
			        "LocaleKey":
			"ESSScheduledTaskRecurrenceType"
			      }
			  }
			}</pre>

参数	本地化键索引	参数值	示例

ECS自定义路由下一跳的类型	ECSRouteNextHopType	• Instance
		• HaVip
		• RouterInterface
		• NetworkInterface
		• VpnGateway
		• IPv6Gateway
		• NatGateway
		• Attachment
			<pre>{
			  "ECSRouteNextHopType": {
			    "Type": "String",
			    "AllowedValues": ["Instance",
			"HaVip", "RouterInterface",
			"NetworkInterface", "VpnGateway",
			"IPv6Gateway", "NatGateway",
			"Attachment"],
			    "AssociationPropertyMetadata": {
			        "LocaleKey":
			"ECSRouteNextHopType"
			      }
			  }
			}</pre>

6.2.3. 资源（Resources）

资源（Resources）用于描述资源栈中每个资源的属性和资源之间的依赖关系。一个资源可以被其他资源引用，也可以在输出（Outputs）中被引用。

资源实体类型

资源实体类型分为普通资源和数据源资源，普通资源可进一步分为阿里云资源和自定义资源，具体如下表所示。

资源实体类型	说明
普通资源（Resource）	• 阿里云资源：用于创建阿里云资源，以及对资源进行更新和删除。阿里云资源类型以 `ALIYUN::` 开头。 • 自定义资源：用于创建自定义逻辑的资源，并可自定义资源如何更新和删除。自定义资源类型以 `Custom::` 开头，或者为 `ALIYUN::ROS::CustomResource` 。 更多信息，请参见概览。
数据源资源（DataSource）	用于查询云服务资源数据。 更多信息，请参见数据源资源。

语法

Resources由资源ID和资源描述组成。资源描述用大括号（{}）括起。如果声明多个资源，用半角逗号（,）分隔开。Resources的语法结构示例代码段如下：

```
"Resources" : {
    "资源1 ID" : {
        "Type" : "资源类型",
        "Condition": "是否创建此资源的条件",
        "Properties" : {
            资源属性描述
        }
    },
    "资源 2 ID" : {
        "Type" : "资源类型",
        "Condition": "是否创建此资源的条件",
        "Properties" : {
            资源属性描述
        }
    }
}
```

参数说明如下：

- 资源ID在模板中具有唯一性。在创建模板其它部分时，可以通过资源ID引用该资源。
- 资源类型（Type）表示正在声明的资源的类型。例如：ALIYUN::ECS::Instance表示阿里云ECS实例。关于ROS支持的所有资源类型列表和详细信息，请参见资源类型索引。
- 资源属性（Properties）是为资源指定的附加选项。例如：必须为每个阿里云ECS实例指定一个Image ID。

示例

```
"Resources" : {
  "ECSInstance" : {
    "Type" : "ALIYUN::ECS::Instance",
    "Properties" : {
      "ImageId" : "m-25l0r****"
    }
  }
}
```

如果资源不需要声明任何属性，可以忽略该资源的属性部分。

属性值可以是文本字符串、字符串列表、布尔值、引用参数或者函数返回值。

- 如果属性值为文本字符串，该值会被双引号（""）括起来。
- 如果属性值为任一类型的字符串列表，则会被方括号（[]）括起来。
- 如果属性值为内部函数或引用的参数，则会被大括号（{}）括起来。

当您将文字、列表、引用参数和函数返回值合并起来取值时，上述规则也适用。
声明不同的属性值类型示例如下：

```
"Properties" : {
    "String" : "string",
    "LiteralList" : [ "value1", "value2" ],
    "Boolean" : "true"
    "ReferenceForOneValue" :  { "Ref" : "ResourceID" } ,
    "FunctionResultWithFunctionParams" : {
        "Fn::Join" : [ "%", [ "Key=", { "Ref" : "SomeParameter" } ] ] }
}
```

DeletionPolicy

在模板中，设置DeletionPolicy属性，可以声明在资源栈被删除时保留某个资源。例如：设置在资源栈删除时保留ECS实例，可按照以下代码段进行声明：

```
"Resources" : {
  "ECSInstance" : {
    "Type" : "ALIYUN::ECS::Instance",
    "Properties" : {
      "ImageId" : "m-25l0r****"
    },
    "DeletionPolicy" : "Retain"
  }
}
```

在本示例中，如果该模板对应的资源栈被删除，则会保留ECSInstance资源。

DependsOn

在模板中，设置DependsOn属性，可以指定特定资源紧跟着另一个资源后创建。为某个资源添加DependsOn属性后，该资源仅在DependsOn属性中指定的资源之后创建。

> **注意** 允许DependsOn依赖的资源Condition为*False*，为*False*时不影响该资源的创建。

用法示例包括以下两种：

- 依赖单个资源：

```
"DependsOn": "ResourceName"
```

- 依赖多个资源：

```
"DependsOn": [
               "ResourceName1",
               "ResourceName2"
             ]
```

如以下代码段所示，WebServer将在DatabaseServer创建成功后才开始创建：

```
{
  "ROSTemplateFormatVersion" : "2015-09-01",
  "Resources" : {
    "WebServer": {
      "Type": "ALIYUN::ECS::Instance",
      "DependsOn": "DatabseServer"
    },
    "DatabseServer": {
      "Type": "ALIYUN::ECS::Instance",
      "Properties": {
        "ImageId" : "m-25l0r****",
        "InstanceType": "ecs.t1.small"
      }
    }
  }
}
```

Condition

在模板中，使用Condition属性可以指定是否需要创建此资源。只有Condition所指定的条件值为*True*时，才会创建此资源。

如以下代码段所示，根据MaxAmount的值判断否创建WebServer:

```
{
  "ROSTemplateFormatVersion" : "2015-09-01",
  "Parameters": {
    "MaxAmount": {
      "Type": "Number",
      "Default": 1
    }
  },
  "Conditions": {
    "CreateWebServer": {"Fn::Not": {"Fn::Equals": [0, {"Ref": "MaxAmount"}]}}
  }
  "Resources" : {
    "WebServer": {
      "Type": "ALIYUN::ECS::InstanceGroup",
      "Condition": "CreateWebServer",
      "Properties": {
        "ImageId" : "m-25l0rc****",
        "InstanceType": "ecs.t1.small"
        "MaxAmount": {"Ref": "MaxAmount"}
      }
    },
    "DatabseServer": {
      "Type": "ALIYUN::ECS::Instance",
      "Properties": {
        "ImageId" : "m-25l0r****",
        "InstanceType": "ecs.t1.small"
      }
    }
  }
}
```

Count

在模板中，为资源指定 `Count` 后，ROS会对模板进行预处理，把该资源展开成多个资源。 操作资源栈时使用处理后的模板。

- 例如：资源名为 `A` ，其 `Count` 值为3。在处理后的模板中，没有 `A` 资源，取而代之的是 `A[0]` 、 `A[1]` 和 `A[2]` 这3个资源。

```
"Resources": {
  "A": {
    "Count": 3,
    ...
  },
  ...
}
```

处理后：

```
"Resources": {
  "A[0]": {
    ...
  },
  "A[1]": {
    ...
  },
  "A[2]": {
    ...
  },
  ...
}
```

> **注意** 如果展开后的资源的名称在原模板中已存在，比如 `A[1]` 在原模板中已经定义，则预处理失败，校验不通过。
> 请避免向已有资源添加 `Count` 属性，因为这会导致资源名发生变化，引发删除操作。

- `Count` 最终结果必须为自然数，仅支持如下函数：
  - Ref（仅引用参数）
  - Fn::FindInMap
  - Fn::Select
  - Fn::Join
  - Fn::Split
  - Fn::Replace
  - Fn::Str
  - Fn::Base64Decode
  - Fn::If
  - Fn::ListMerge
  - Fn::GetJsonValue
  - Fn::MergeMapToList
  - Fn::SelectMapList

- Fn::Add
- Fn::Avg
- Fn::Str
- Fn::Calculate
- Fn::Max
- Fn::Min
- Fn::Jq
- Fn::Length
- Fn::GetAZs
- Fn::Index
- Fn::FormatTime
- Fn::MarketplaceImage

- 处理后的模板资源总数需要符合限制，目前最多支持300个。
- 指定了 `Count` 的资源的属性(`Properties`)中可以使用伪参数 `ALIYUN::Index` ，在预处理的时候会被替换为相应的数值。例如， `A[0]` 使用的 `ALIYUN::Index` 会被替换为0， `A[1]` 使用的 `ALIYUN::Index` 会被替换为1。模板其他地方不能使用 `ALIYUN::Index` 。
- 如果指定了 `Count` 的资源出现在 `DependsOn` 中，会被展开。

  ```
  "DependsOn": "A"
  ```

  处理后：

  ```
  "DependsOn": ["A[0]", "A[1]", "A[2]"]
  ```

- 如果指定了 `Count` 的资源出现在函数 `Ref` 、 `Fn::GetAtt` 中，会被展开。

  ```
  {
    "Ref": "A"
  }
  {
    "Fn::GetAtt": ["A", "PropertyName"]
  }
  ```

  处理后：

```
[
  {
    "Ref": "A[0]"
  },
  {
    "Ref": "A[1]"
  },
  {
    "Ref": "A[2]"
  }
]
[
  {
    "Fn::GetAtt": [
      "A[0]",
      "PropertyName"
    ]
  },
  {
    "Fn::GetAtt": [
      "A[1]",
      "PropertyName"
    ]
  },
  {
    "Fn::GetAtt": [
      "A[2]",
      "PropertyName"
    ]
  }
]
```

如果多个资源使用 `Count` ，并且存在引用关系。建议与 `Fn::Select` 以及 `ALIYUN::Index` 联合使用。

```
{
  "Fn::Select": [
    {
      "Ref": "ALIYUN::Index"
    },
    {
      "Ref": "A"
    }
  ]
}
```

以 `A` 为例， `B` 引用了 `A` ，且 `B` 的 `Count` 为2，转换后 `B[0]` 、 `B[1]` 中部分表达式分别如下：

```
{
  "Ref": "A[0]"
}
{
  "Ref": "A[1]"
}
```

- 如果资源指定了 `Count` 属性，则 `DependsOn` 属性也支持表达式。 `DependsOn` 属性的表达式可以使用的函数与 `Count` 属性一致，可以使用 `ALIYUN::Index` 伪参数。表达式的格式与允许的计算结果如下表所示。

格式	示例	允许的计算结果
字典。	`{ "Fn::Split": [",", "Server1,Server2"] }`	null。 如果为null值，则会移除DependsOn属性。
		字符串。 ○ 如果是空字符串，则会移除DependsOn属性。 ○ 如果是非空字符串，则必须是有效的资源名称。
		列表。 ○ 如果列表项为null或空字符串，则丢弃该列表项。 ○ 如果列表为空，则会移除DependsOn属性。 ○ 如果列表不为空，则每一项必须是字符串，且为有效的资源名称。
列表。	`[ "Server0", { "Fn::Split": [",", "Server1,Server2"] } ]`	列表项可以是字典或字符串，不同列表项允许的计算结果不同，具体如下： ○ 如果列表项为字典，则会进行表达式计算，允许的计算结果如下： ■ 如果为null值或空字符串，则丢弃该项。 ■ 否则，必须是字符串，且为有效的资源名称。 ○ 如果列表项为字符串，则必须是有效的资源名称。 **说明** 如果列表为空，则会移除DependsOn属性。
字符串。	`Server0`	字符串。 不会进行额外的处理。

如下示例模板用于控制Count资源单并发数。参数Count表示要创建的资源的数量，参数ParallelCount代表最大并发创建数。

```
{
  "ROSTemplateFormatVersion": "2015-09-01",
  "Parameters": {
    "Count": {
      "Type": "Number"
    },
    "ParallelCount": {
      "Type": "Number"
    }
  },
  "Resources": {
    "WaitConditionHandle": {
      "Type": "ALIYUN::ROS::WaitConditionHandle",
      "Count": {
        "Ref": "Count"
      },
      "DependsOn": {
        "Fn::Select": [
          {
            "Fn::Calculate": [
              "1-{0}//{1}",
              0,
              [
                {
                  "Fn::Min": [
                    {
                      "Ref": "ALIYUN::Index"
                    },
                    {
                      "Ref": "ParallelCount"
                    }
                  ]
                },
                {
                  "Ref": "ParallelCount"
                }
              ]
            ]
          },
          [
            {
              "Fn::Replace": [
                {
                  "index": {
                    "Fn::Calculate": [
                      "{0}-{1}",
                      0,
                      [
                        {
                          "Ref": "ALIYUN::Index"
                        },
                        {
                          "Ref": "ParallelCount"
                        }
```

```
                    ]
                  ]
                }
              },
              "WaitConditionHandle[index]"
            ]
          }
        ]
      ]
    }
  }
}
```

当Count和ParallelCount取值不同时，处理后的模板也有差异，具体如下：

- 若设置Count为3，ParallelCount为1，则处理后的模板为：

```
{
  "ROSTemplateFormatVersion": "2015-09-01",
  "Resources": {
    "WaitConditionHandle[0]": {
      "Type": "ALIYUN::ROS::WaitConditionHandle"
    },
    "WaitConditionHandle[1]": {
      "Type": "ALIYUN::ROS::WaitConditionHandle",
      "DependsOn": "WaitConditionHandle[0]"
    },
    "WaitConditionHandle[2]": {
      "Type": "ALIYUN::ROS::WaitConditionHandle",
      "DependsOn": "WaitConditionHandle[1]"
    }
  },
  "Parameters": {
    "Count": {
      "Type": "Number"
    },
    "ParallelCount": {
      "Type": "Number"
    }
  }
}
```

- 若设置Count为5，ParallelCount为2，则处理后的模板为：

```
{
  "ROSTemplateFormatVersion": "2015-09-01",
  "Resources": {
    "WaitConditionHandle[0]": {
      "Type": "ALIYUN::ROS::WaitConditionHandle"
    },
    "WaitConditionHandle[1]": {
      "Type": "ALIYUN::ROS::WaitConditionHandle"
    },
    "WaitConditionHandle[2]": {
      "Type": "ALIYUN::ROS::WaitConditionHandle",
      "DependsOn": "WaitConditionHandle[0]"
    },
    "WaitConditionHandle[3]": {
      "Type": "ALIYUN::ROS::WaitConditionHandle",
      "DependsOn": "WaitConditionHandle[1]"
    },
    "WaitConditionHandle[4]": {
      "Type": "ALIYUN::ROS::WaitConditionHandle",
      "DependsOn": "WaitConditionHandle[2]"
    }
  },
  "Parameters": {
    "Count": {
      "Type": "Number"
    },
    "ParallelCount": {
      "Type": "Number"
    }
  }
}
```

- 若设置Count为5，ParallelCount为3，则处理后的模板为：

```
{
  "ROSTemplateFormatVersion": "2015-09-01",
  "Resources": {
    "WaitConditionHandle[0]": {
      "Type": "ALIYUN::ROS::WaitConditionHandle"
    },
    "WaitConditionHandle[1]": {
      "Type": "ALIYUN::ROS::WaitConditionHandle"
    },
    "WaitConditionHandle[2]": {
      "Type": "ALIYUN::ROS::WaitConditionHandle"
    },
    "WaitConditionHandle[3]": {
      "Type": "ALIYUN::ROS::WaitConditionHandle",
      "DependsOn": "WaitConditionHandle[0]"
    },
    "WaitConditionHandle[4]": {
      "Type": "ALIYUN::ROS::WaitConditionHandle",
      "DependsOn": "WaitConditionHandle[1]"
    }
  },
  "Parameters": {
    "Count": {
      "Type": "Number"
    },
    "ParallelCount": {
      "Type": "Number"
    }
  }
}
```

Count的示例模板如下。示例模板中创建了一组EIP和同等数量的ECS，并把EIP与ECS逐一绑定。

```
{
  "ROSTemplateFormatVersion": "2015-09-01",
  "Parameters": {
    "Count": {
      "Type": "Number"
    }
  },
  "Resources": {
    "Eip": {
      "Type": "ALIYUN::VPC::EIP",
      "Count": {
        "Ref": "Count"
      },
      "Properties": {
        "Bandwidth": 5
      }
    },
    "Servers": {
      "Type": "ALIYUN::ECS::InstanceGroup",
```

```
      "Properties": {
        "MinAmount": {
          "Ref": "Count"
        },
        "MaxAmount": {
          "Ref": "Count"
        },
        "AllocatePublicIP": false,
        ...
      }
    },
    "EipBind": {
      "Type": "ALIYUN::VPC::EIPAssociation",
      "Count": {
        "Ref": "Count"
      },
      "Properties": {
        "InstanceId": {
          "Fn::Select": [
            {
              "Ref": "ALIYUN::Index"
            },
            {
              "Fn::GetAtt": [
                "Servers",
                "InstanceIds"
              ]
            }
          ]
        },
        "AllocationId": {
          "Fn::Select": [
            {
              "Ref": "ALIYUN::Index"
            },
            {
              "Ref": "Eip"
            }
          ]
        }
      }
    }
  },
  "Outputs": {
    "InstanceIds": {
      "Value": {
        "Fn::GetAtt": [
          "Servers",
          "InstanceIds"
        ]
      }
    },
    "AllocationIds": {
      "Value": {
        "Ref": "Eip"
```

```
      }
    },
    "EipAddresses": {
      "Value": {
        "Fn::GetAtt": [
          "Eip",
          "EipAddress"
        ]
      }
    }
  }
}
```

处理后：

```
{
  "ROSTemplateFormatVersion": "2015-09-01",
  "Parameters": {
    "Count": {
      "Type": "Number"
    }
  },
  "Resources": {
    "Eip[0]": {
      "Type": "ALIYUN::VPC::EIP",
      "Properties": {
        "Bandwidth": 5
      }
    },
    "Eip[1]": {
      "Type": "ALIYUN::VPC::EIP",
      "Properties": {
        "Bandwidth": 5
      }
    },
    "Servers": {
      "Type": "ALIYUN::ECS::InstanceGroup",
      "Properties": {
        "MinAmount": {
          "Ref": "Count"
        },
        "MaxAmount": {
          "Ref": "Count"
        },
        "AllocatePublicIP": false,
        ...
      }
    },
    "EipBind[0]": {
      "Type": "ALIYUN::VPC::EIPAssociation",
      "Properties": {
        "InstanceId": {
          "Fn::Select": [
```

```
            0,
            {
              "Fn::GetAtt": [
                "Servers",
                "InstanceIds"
              ]
            }
          ]
        },
        "AllocationId": {
          "Ref": "Eip[0]"
        }
      }
    },
    "EipBind[1]": {
      "Type": "ALIYUN::VPC::EIPAssociation",
      "Properties": {
        "InstanceId": {
          "Fn::Select": [
            1,
            {
              "Fn::GetAtt": [
                "Servers",
                "InstanceIds"
              ]
            }
          ]
        },
        "AllocationId": {
          "Ref": "Eip[1]"
        }
      }
    }
  },
  "Outputs": {
    "InstanceIds": {
      "Value": {
        "Fn::GetAtt": [
          "Servers",
          "InstanceIds"
        ]
      }
    },
    "AllocationIds": {
      "Value": [
        {
          "Ref": "Eip[0]"
        },
        {
          "Ref": "Eip[1]"
        }
      ]
    },
    "EipAddresses": {
      "Value": [
```

```
        "Value": [
          {
            "Fn::GetAtt": [
              "Eip[0]",
              "EipAddress"
            ]
          },
          {
            "Fn::GetAtt": [
              "Eip[1]",
              "EipAddress"
            ]
          }
        ]
      }
    }
}
```

资源声明示例

典型的资源声明示例如下：

```
"Resources" : {
    "WebServer": {
      "Type": "ALIYUN::ECS::Instance",
      "Properties": {
        "ImageId" : "m-25l0r****",
        "InstanceType": "ecs.t1.small",
        "SecurityGroupId": "sg-25zwc****",
        "ZoneId": "cn-beijing-b",
        "Tags": [{
            "Key": "Department1",
            "Value": "HumanResource"
        },{
            "Key": "Department2",
            "Value": "Finance"
        }
        ]
      }
    },
    "ScalingConfiguration": {
      "Type": "ALIYUN::ESS::ScalingConfiguration",
      "Properties": {
        "ImageId": "ubuntu_14_04_64_20G_aliaegis_2015****.vhd",
        "InstanceType": "ecs.t1.small",
        "InstanceId": "i-25xhh****",
        "InternetChargeType": "PayByTraffic",
        "InternetMaxBandwidthIn": 1,
        "InternetMaxBandwidthOut": 20,
        "SystemDisk_Category": "cloud",
        "ScalingGroupId": "bwhtvpcBcKYac9fe3vd0****",
        "SecurityGroupId": "sg-25zwc****",
        "DiskMappings": [
            {
                "Size": 10
            },
            {
                "Category": "cloud",
                "Size": 10
            }
        ]
      }
    }
}
```

6.2.4. 输出（Outputs）

在输出（Outputs）中，定义在调用查询资源栈接口时返回的值。例如，定义ECS实例ID的输出，然后可在调用查询资源栈的接口时，查看该实例ID。

语法

Outputs由输出ID和输出描述组成。所有输出描述都被括在大括号（{}）里。如果声明多个输出项，则用半角逗号（,）分隔开。每一个输出项支持以数组形式输出多个值。请参见以下Outputs的语法结构示例代码：

```
"Outputs" : {
  "输出1 ID" : {
    "Description" : "输出的描述",
    "Condition": "是否输出此资源属性的条件",
    "Value" : "输出值的表达式"
  },
  "输出2 ID" : {
    "Description" : "输出的描述",
    "Condition": "是否输出此资源属性的条件",
    "Value" : [
      "输出值的表达式1",
      "输出值的表达式2",
      ...
    ]
  }
}
```

- 输出ID：输出项的标识符，在模板中具有唯一性。
- Description（可选）：对输出值的描述。
- Value（必需）：在调用查询资源栈接口时，返回的属性值。
- Condition（可选）：使用Condition属性可以指定是否需要创建某个资源和输出资源的信息。当Condition所指定的条件值为true时，才创建此资源和输出资源信息。

示例

在以下示例中，输出部分有 2 个输出项。第一个输出资源ID为WebServer的InstanceId属性，第二个输出资源ID为WebServer的PublicIp和PrivateIp属性。

```
"Outputs": {
  "InstanceId": {
    "Value" : {"Fn::GetAtt": ["WebServer", "InstanceId"]}
  },
  "PublicIp & PrivateIp": {
    "Value" : [
      {"Fn::GetAtt": ["WebServer", "PublicIp"]},
      {"Fn::GetAtt": ["WebServer", "PrivateIp"]}
    ]
  }
}
```

在以下示例中，根据MaxAmount的值判断是否创建WebServer。

```
{
  "ROSTemplateFormatVersion" : "2015-09-01",
  "Parameters": {
    "MaxAmount": {      "Type": "Number",
      "Default": 1
    }
  },
  "Conditions": {
    "CreateWebServer": {"Fn::Not": {"Fn::Equals": [0, {"Ref": "MaxAmount"}]}}
  }
  "Resources" : {
    "WebServer": {
      "Type": "ALIYUN::ECS::InstanceGroup",
      "Condition": "CreateWebServer",
      "Properties": {
        "ImageId" : "m-25l0r****",
        "InstanceType": "ecs.t1.small"
        "MaxAmount": {"Ref": "MaxAmount"}
      }
    }
  }
  "Outputs": {
    "WebServerIP": {
      "Condition": "CreateWebServer",
      "Value": {
        "Fn::GetAtt": ["WebServer", "PublicIps"]
      }
    }
  }
}
```

6.2.5. 函数（Functions）

资源编排服务提供多个内部函数，帮助您管理资源栈。您可以在定义资源（Resources）和输出（Outputs）时，使用内部函数。
可在资源栈模板中使用的内部函数包括：Fn::Str、Fn::Base64Encode、Fn::Base64Decode、Fn::FindInMap、Fn::GetAtt、Fn::Join、Fn::Sub、Fn::Select、Ref、Fn::GetAZs、Fn::Replace、Fn::Split、Fn::Equals、Fn::And、Fn::Or、Fn::Not、Fn::Index、Fn::If、Fn::Length、Fn::ListMerge、Fn::GetJsonValue、Fn::MergeMapToList、Fn::Avg、Fn::SelectMapList、Fn::Add、Fn::Calculate、Fn::Max、Fn::Min、Fn::GetStackOutput、Fn::Jq、Fn::FormatTime、Fn::MarketplaceImage和Fn::Any。

Fn::Str

内部函数Fn::Str返回输入数字的字符串结果。

- 声明

```
"Fn::Str" : toString
```

- 参数
  `toString`：需要转换的Number/Integer类型的值。
- 返回值
  该值的字符串类型。

- 示例

```
{"Fn::Str": 123456}
```

返回：`"123456"`。

Fn::Base64Encode

内部函数Fn::Base64Encode返回输入字符串的Base64编码结果。

- 声明

```
"Fn::Base64Encode": "stringToEncode"
```

- 参数
  `stringToEncode`：转换成Base64的字符串。
- 返回值
  用Base64表示的原始字符串。
- 示例

```
{"Fn::Base64Encode": "string to encode"}
```

返回：`c3RyaW5nIHRvIGVuY29kZQ==`。

Fn::Base64Decode

内部函数Fn::Base64Decode返回Base64编码的字符串解码的结果。

- 声明

```
{"Fn::Base64Decode": "stringToEncode"}
```

- 参数
  `stringToDecode`：将Base64编码的字符串解码。
- 返回值
  字符串。
- 示例

```
{"Fn::Base64Decode": "c3RyaW5nIHRvIGVuY29kZQ=="}
```

返回：`string to encode`。

Fn::FindInMap

内部函数Fn::FindInMap返回与Mappings声明的双层映射中的键对应的值。

- 声明

```
"Fn::FindInMap": ["MapName", "TopLevelKey", "SecondLevelKey"]
```

- 参数
  - `MapName`：Mappings中所声明映射的ID，包含键和值。
  - `TopLevelKey`：第一级键，其值是一个键/值对列表。
  - `SecondLevelKey`：第二级键，其值是一个字符串或者数字。
- 返回值
  分配给SecondLevelKey的值。

- 示例
  在创建名为WebServer的资源时，需要指定ImageId属性。在Mappings中，描述根据区域区分的ImageId映射。在Parameters中，描述需要用户指定的区域。Fn::FindInMap会根据用户指定的区域，在RegionMap中查找对应的ImageId映射，然后在映射中找到对应的ImageId。
  - MapName可按照个人意愿设置。在本示例中为 `"RegionMap"` 。
  - TopLevelKey设置为创建资源栈的地区。在本示例中，通过 `{"Ref" : "regionParam"}` 确定。
  - SecondLevelKey设置为所需的架构。在本示例中为 `"32"` 。

```
{
  "ROSTemplateFormatVersion": "2015-09-01",
  "Parameters": {
    "regionParam": {
      "Description": "选择创建ECS的区域",
      "Type": "String",
      "AllowedValues": [
        "hangzhou",
        "beijing"
      ]
    }
  },
  "Mappings": {
    "RegionMap": {
      "hangzhou": {
        "32": "m-25l0rcfjo",
        "64": "m-25l0rcfj1"
      },
      "beijing": {
        "32": "m-25l0rcfj2",
        "64": "m-25l0rcfj3"
      }
    }
  },
  "Resources": {
    "WebServer": {
      "Type": "ALIYUN::ECS::Instance",
      "Properties": {
        "ImageId": {
          "Fn::FindInMap": [
            "RegionMap",
            {"Ref": "regionParam"},
            "32"
          ]
        },
        "InstanceType": "ecs.t1.small",
        "SecurityGroupId": "sg-25zwc****",
        "ZoneId": "cn-beijing-b",
        "Tags": [
          {
            "Key": "key1",
            "Value": "value1"
          },
          {
            "Key": "key2",
            "Value": "value2"
          }
        ]
      }
    }
  }
}
```

- 支持的函数

- Fn::FindInMap
- Ref

Fn::GetAtt

内部函数Fn::GetAtt返回模板中的资源的属性值。

- 场景一：Fn::GetAtt函数用于资源栈
  - 声明

    ```
    "Fn::GetAtt": ["resourceID", "attributeName"]
    ```

  - 参数
    - `resourceID`：资源栈中资源的ID。
    - `attributeName`：资源栈中资源的属性名称。
  - 返回值
    属性值。
  - 示例
    返回Resource ID为MyEcsInstance的ImageId属性。

    ```
    {"Fn::GetAtt" : ["MyEcsInstance" , "ImageID"]}
    ```

- 场景二：Fn::GetAtt函数用于嵌套资源栈
  - 声明

    ```
    "Fn::GetAtt": ["nestedStack", "Outputs.attributeName"]
    ```

  - 参数
    - `nestedStack`：嵌套资源栈的名称。
    - `Outputs.attributeName`：`Outputs.`前缀固定，`attributeName`为嵌套资源栈输出的属性名称。
  - 返回值
    嵌套资源栈的输出值。
  - 示例
    关于示例的更多信息，请参见ALIYUN::ROS::Stack。

Fn::Join

内部函数Fn::Join将一组值连接起来，用特定分隔符隔开。

> **说明** Fn::Sub和Fn::Join均可用于拼接字符串。Fn::Join只能把多个字符串拼接成一个字符串，而Fn::Sub可以把多个字符串或数字拼接成字符串，更为简洁和直观，因此推荐您使用Fn::Sub。

- 声明

  ```
  {"Fn::Join": ["delimiter", ["string1", "string2", ... ]]}
  ```

- 参数
  - `delimiter`：分隔符。分隔符可以为空，表示将所有的值直接连接起来。
  - `["string1", "string2", ...]`：被连接起来的值列表。

- 返回值
  被连接起来的字符串。
- 示例

```
{"Fn::Join": [ ",", ["a", "b", "c"]]}
```

  返回： `"a,b,c"` 。
- 支持的函数
  - Fn::Base64Encode
  - Fn::GetAtt
  - Fn::Join
  - Fn::Select
  - Ref

Fn::Sub

内部函数Fn::Sub用于将输入字符串中的变量替换为您指定的值。

- 声明

```
{"Fn::Sub": [ String, { Var1Name: Var1Value, Var2Name: Var2Value, ... } ] }
```

- 参数
  - String
    一个能进行变量替换的字符串，ROS在运行时会将这些变量替换为指定值。以 `${VarName}` 形式编写变量，变量可以是模板参数、伪参数、资源逻辑ID、资源属性或键值映射中的变量等。如果您仅指定模板参数、伪参数、资源逻辑ID和资源属性，则不需要指定键值映射。
    如果您指定模板参数名、伪参数或资源逻辑ID（如 `${MyParameter}` ），则ROS返回的值与您使用 `Ref` 内部函数时返回的值相同。如果您指定资源属性（如 `${MyInstance.InstanceId}` ），则ROS返回的值与您使用 `Fn::GetAtt` 内部函数时返回的值相同。
    要按字面书写美元符号（$）和大括号（{}），请在左大括号后面添加感叹号（!），
    如 `${!Literal}` 。ROS将该文本解析为 `${Literal}` 。
  - VarName
    String参数中包含的变量的名称。
  - VarValue
    ROS在运行时要将关联的变量名称替换为的值。

  如果不需要使用键值映射，可使用以下声明方式：

```
{"Fn::Sub": String}
```

- 返回值
  ROS返回原始字符串，并替换所有变量的值。
- 示例

```
{
  "ROSTemplateFormatVersion": "2015-09-01",
  "Parameters": {
    "VpcName": {
      "Type": "String",
      "Default": "vpc"
    }
  },
  "Resources": {
    "Vpc": {
      "Type": "ALIYUN::ECS::VPC",
      "Properties": {
        "VpcName": {
          "Ref": "VpcName"
        },
        "CidrBlock": "10.0.XX.XX"
      }
    }
  },
  "Outputs": {
    "Pseudo": {
      "Value": {
        "Fn::Sub": [
          "Var1: ${Var1}, Var2: ${Var2}, StackName: ${ALIYUN::StackName}, Region: ${ALIYU
N::Region}",
          {
            "Var1": "Var1Value",
            "Var2": "Var2Value"
          }
        ]
      }
    },
    "VpcId": {
      "Value": {
        "Fn::Sub": "资源的返回值: ${Vpc.VpcId}, 资源ID: ${Vpc}"
      }
    }
  }
}
```

返回

```
Var1: Var1Value, Var2: Var2Value, StackName: SubTest, Region: cn-hangzhou
资源的返回值: vpc-bp11eu7avmtvr37hl****, 资源ID: vpc-bp11eu7avmtvr37hl****
```

Fn::Select

内部函数Fn::Select通过索引返回列表或字典中的数据。

- 声明

- 获取列表（数组）中的数据
  - 根据索引获取单个数据

```
{"Fn::Select": ["index", ["value1", "value2", ...]]}
```

  - 获取多个数据

```
{"Fn::Select": ["start:stop", ["value1", "value2", ...]]}
```

```
{"Fn::Select": ["start:stop:step", ["value1", "value2", ... ]]}
```

- 获取字典（映射表）某个键对应的值：

```
{"Fn::Select": ["key", {"key1": "value1", "key2": "value2", ...}]}
```

- 参数
  - `index`：待检索数据的索引。索引是0到N-1之间或-N到-1之间的某个值（其中N代表列表中元素的数量），负数表示从右往左第几个元素。如果索引不在取值范围内，则返回空字符串。
  - `start`、`stop`和`step`：根据指定的起始位置和终止位置获取列表中的元素，可以指定步长，即每隔step-1个元素取一个元素，返回一个列表。
    - `start:stop`：start、stop取值同index，start不填时默认为0，stop不填时默认为N。返回列表中第start+1到第stop个元素组成的列表，如果二者取值不在取值范围内则返回空列表。
    - `start:stop:step`：step不填时默认为1，如果step为负数，则start表示的元素的索引应大于stop表示的元素的索引，从第start个元素到第stop+1元素，每隔-step-1个元素取一个元素，返回一个列表。
  - `key`：字典的某个键，返回键对应的值，如果字典中不存在该键则返回空字符串。
- 返回值
  选定的数据。
- 示例
  - 从列表或数组中取值：

```
{"Fn::Select": ["1", ["apples", "grapes", "oranges", "mangoes"]]}
```

返回：`"grapes"`。

```
{"Fn::Select": ["1:3", [1, 2, 3, 4, 5]]}
```

返回：`[2, 3]`

```
{"Fn::Select": ["::2", [1, 2, 3, 4, 5]]}
```

返回：`[1, 3, 5]`

```
{"Fn::Select": ["5:0:-2", [1, 2, 3, 4, 5]]}
```

返回：`[5, 3]`

- 获取字典某个键对应的值：

```
{"Fn::Select": ["key1", {"key1": "grapes", "key2": "mangoes"}]}
```

返回：`"grapes"`。

- 如果数据元列表是一个CommaDelimitedList：

```
"Parameters": {
  "userParam": {
    "Type": "CommaDelimitedList",
      "Default": "10.0.0.1, 10.0.0.2, 10.0.0.3"
  }
}
"Resources": {
  "resourceID": {
      "Properties": {
          "CidrBlock": {"Fn::Select": ["0", {"Ref": "userParam"}]}
    }
  }
}
```

返回：`10.0.0.1`

- 支持的函数
  对于Fn::Select索引值，您可以使用Ref函数。
  对于对象的Fn::Select列表，您可以使用以下函数：
  - Fn::Base64Encode
  - Fn::FindInMap
  - Fn::GetAtt
  - Fn::Join
  - Fn::Select
  - Ref

Ref

内部函数Ref返回指定参数或资源的值。

如果指定参数是Resource ID，则返回资源的值。否则系统将认为指定参数是参数，将尝试返回参数的值。

- 声明

```
"Ref": "logicalName"
```

- 参数
  `logicalName`：要引用的资源或参数的逻辑名称。
- 返回值
  资源的值或者参数的值。
- 示例
  使用Ref函数指定regionParam作为WebServer的RegionMap的区域参数。

```
{
  "ROSTemplateFormatVersion": "2015-09-01",
  "Parameters": {
    "regionParam": {
      "Description": "选择创建ECS的区域",
      "Type": "String",
      "AllowedValues": [
        "hangzhou",
        "beijing"
      ]
    }
  },
  "Mappings": {
    "RegionMap": {
      "hangzhou": {
        "32": "m-25l0rcfjo",
        "64": "m-25l0rcfj1"
      },
      "beijing": {
        "32": "m-25l0rcfj2",
        "64": "m-25l0rcfj3"
      }
    }
  },
  "Resources": {
    "WebServer": {
      "Type": "ALIYUN::ECS::Instance",
      "Properties": {
        "ImageId": {
          "Fn::FindInMap": [
            "RegionMap",
            {"Ref": "regionParam"},
            "32"
          ]
        },
        "InstanceType": "ecs.t1.small",
        "SecurityGroupId": "sg-25zwc****",
        "ZoneId": "cn-beijing-b",
        "Tags": [
          {
            "Key": "tiantt",
            "Value": "ros"
          },
          {
            "Key": "tiantt1",
            "Value": "ros1"
          }
        ]
      }
    }
  }
}
```

- 支持的函数

不能在Ref函数中使用任何函数。必须指定为资源逻辑ID的字符串。

Fn::GetAZs

内部函数Fn::GetAZs返回指定Region的可用区列表。

> **说明** 该函数只适用于ECS和VPC类型的资源。

- 声明

```
"Fn::GetAZs": "region"
```

- 参数

`region`：region ID。

- 返回值

指定Region下的可用区列表。

- 示例

在指定Region的第一个可用区内创建一个ECS实例。

```
{
  "ROSTemplateFormatVersion" : "2015-09-01",
  "Resources" : {
    "WebServer": {
      "Type": "ALIYUN::ECS::Instance",
      "Properties": {
        "ImageId": "centos7u2_64_40G_cloudinit_2016072****",
        "InstanceType": "ecs.n1.tiny",
        "SecurityGroupId": "sg-2zedcm7ep5quses0****",
        "Password": "Ros1****",
        "AllocatePublicIP": true,
        "InternetChargeType": "PayByTraffic",
        "InternetMaxBandwidthIn": 100,
        "InternetMaxBandwidthOut": 100,
        "SystemDiskCategory": "cloud_efficiency",
        "IoOptimized": "optimized",
        "ZoneId": {"Fn::Select": ["0", {"Fn::GetAZs": {"Ref": "ALIYUN::Region"}}]}
      }
    }
  },
  "Outputs": {
    "InstanceId": {
         "Value": {"Fn::GetAtt": ["WebServer","InstanceId"]}
    },
    "PublicIp": {
         "Value": {"Fn::GetAtt": ["WebServer","PublicIp"]}
    }
  }
}
```

- 支持的函数
  - Fn::Base64Encode
  - Fn::FindInMap
  - Fn::GetAtt

- Fn::Join
- Fn::Select
- Ref

Fn::Replace

内部函数Fn::Replace将字符串中指定子字符串用新字符串替换。

- 声明

```
{"Fn::Replace": [{"object_key": "object_value"}, "object_string"]}
```

- 参数
  - `object_key`：将要被替换的字符串。
  - `object_value`：将要替换成的最终字符串。
  - `object_string`：包含被替换`object_key`的字符串。
- 返回值
  被替换后的字符串。
- 示例
  所指定脚本中的print替换成echo。

```
{
  "ROSTemplateFormatVersion" : "2015-09-01",
  "Resources" : {
    "WebServer": {
      "Type": "ALIYUN::ECS::Instance",
      "Properties": {
        "ImageId" : "centos_7_2_64_40G_base_20170222****",
        "InstanceType": "ecs.n1.medium",
        "SecurityGroupId": "sg-94q49****",
        "Password": "MytestPassword****",
        "IoOptimized": "optimized",
        "VSwitchId": "vsw-94vdv****",
        "VpcId": "vpc-949uz****",
        "SystemDiskCategory": "cloud_ssd",
        "UserData": {"Fn::Replace": [{"print": "echo"},
         {"Fn::Join": ["", [
             "#!/bin/sh\n",
             "mkdir ~/test_ros\n",
             "print hello > ~/1.txt\n"
         ]]}]}
      }
    }
  },
  "Outputs": {
    "InstanceId": {
         "Value" : {"Fn::GetAtt": ["WebServer","InstanceId"]}
    },
    "PublicIp": {
         "Value" : {"Fn::GetAtt": ["WebServer","PublicIp"]}
    }
  }
}
```

- 支持的函数
  - Fn::Base64Encode
  - Fn::GetAtt
  - Fn::Join
  - Fn::Select
  - Ref

Fn::Split

内部函数Fn::Split通过指定分隔符对字符串进行切片，并返回所有切片组成的列表。

- 声明

```
"Fn::Split": ["delim",  "original_string"]
```

- 参数
  - `delim`：分隔符，例如：英文逗号（,）、半角分号（;）、换行符（\n）、缩进（\t）等。
  - `original_string`：将要被切片的字符串。
- 返回值

返回切片后所有字符串组成的列表。

- 示例
  - 如果数据元列表是一个数组：

```
{"Fn::Split": [";", "foo; bar; achoo"]}
```

返回： `["foo", " bar", "achoo "]` 。

  - 使用Fn::Split对InstanceIds进行切片：

```
{
  "Parameters": {
    "InstanceIds": {
      "Type": "String",
      "Default": "instane1_id,instance2_id,instance2_id"
    }
  },
  "Resources": {
    "resourceID": {
      "Type": "ALIYUN::SLB::BackendServerAttachment",
      "Properties": {
        "BackendServerList": {
          "Fn::Split": [
            ",",
            {
              "Ref": "InstanceIds"
            }
          ]
        }
      }
    }
  }
}
```

- 支持的函数
  - Fn::Base64Encode
  - Fn::FindInMap
  - Fn::GetAtt
  - Fn::Join
  - Fn::Select
  - Fn::Replace
  - Fn::GetAZs
  - Fn::If
  - Ref

Fn::Equals

比较两个值是否相等。如果两个值相等，则返回true；如果不相等，则返回false。

- 声明

```
{"Fn::Equals": ["value_1", "value_2"]}
```

- 参数
  `value`：要比较的任意类型的值。
- 返回值
  true或false。
- 示例
  在Conditions中使用Fn::Equals定义条件。

```
{
  "ROSTemplateFormatVersion": "2015-09-01",
  "Parameters": {
    "EnvType": {
      "Default": "pre",
      "Type": "String"
    }
  },
  "Conditions": {
    "TestEqualsCond": {
      "Fn::Equals": [
        "prod",
        {"Ref": "EnvType"}
      ]
    }
  }
}
```

- 支持的函数
  - Fn::Or
  - Fn::Not
  - Fn::Equals
  - Fn::FindInMap
  - Fn::And
  - Ref

Fn::And

代表AND运算符，最少包含两个条件。如果所有指定条件计算为true，则返回true；如果任意条件计算为false，则返回false。

- 声明

```
{"Fn::And": ["condition", {...}]}
```

- 参数
  `condition`：计算为true或false的条件。
- 返回值
  true或false。
- 示例
  在Conditions中使用Fn::And定义一个条件：

```
{
  "ROSTemplateFormatVersion" : "2015-09-01",
  "Parameters":{
    "EnvType":{
      "Default":"pre",
      "Type":"String"
    }
  },
  "Conditions": {
    "TestEqualsCond": {"Fn::Equals": ["prod", {"Ref": "EnvType"}]},
    "TestAndCond": {"Fn::And": ["TestEqualsCond", {"Fn::Equals": ["pre", {"Ref": "EnvType
"}]}]}
  }
}
```

- 支持的函数
  - Fn::Or
  - Fn::Not
  - Fn::Equals
  - Fn::FindInMap
  - Fn::And
  - Ref

Fn::Or

代表OR运算符，最少包含两个条件。如果任意一个指定条件计算为true，则返回true；如果所有条件都计算为false，则返回false。

- 声明

```
{"Fn::Or": ["condition", {...}]}
```

- 参数
  `condition`：计算为true或false的条件。
- 返回值
  true或false。
- 示例
  在Conditions中使用Fn::Or定义一个条件。

```
{
  "ROSTemplateFormatVersion" : "2015-09-01",
  "Parameters":{
    "EnvType":{
      "Default":"pre",
      "Type":"String"
    }
  },
  "Conditions": {
    "TestEqualsCond": {"Fn::Equals": ["prod", {"Ref": "EnvType"}]},
    "TestOrCond": {"Fn::Or": ["TestEqualsCond", {"Fn::Equals": ["pre", {"Ref": "EnvType"}
]}]}
  }
}
```

- 支持的函数
  - Fn::Or
  - Fn::Not
  - Fn::Equals
  - Fn::FindInMap
  - Fn::And
  - Ref

Fn::Not

代表NOT运算符。对计算为false的条件，返回true；对计算为true的条件，返回false。

- 声明

```
{"Fn::Not": "condition"}
```

- 参数
  `condition`：计算为true或false的条件。
- 返回值
  true或false。
- 示例
  在Conditions中使用Fn::Not定义一个条件。

```
{
  "ROSTemplateFormatVersion" : "2015-09-01",
  "Parameters":{
    "EnvType":{
      "Default":"pre",
      "Type":"String"
    }
  },
  "Conditions": {
    "TestNotCond": {"Fn::Not": {"Fn::Equals": ["pre", {"Ref": "EnvType"}]}}
  }
}
```

- 支持的函数
  - Fn::Or

- Fn::Not
- Fn::Equals
- Fn::FindInMap
- Fn::And
- Ref

Fn::Index

用于查找列表中某个元素的索引。

- 声明

```
{"Fn::Index": ["item", [ ... ]]}
```

- 参数

`item`：列表中的元素。

- 返回值

列表中元素的索引，不存在则返回空。

- 示例

```
{
  "ROSTemplateFormatVersion": "2015-09-01",
  "Parameters": {
    "ZoneIds": {
      "Type": "Json",
      "Default": ["cn-beijing-a", "cn-beijing-b", "cn-beijing-f"]
    },
    "ZoneId": {
      "Type": "String",
      "Default": "cn-beijing-b"
    }
  },
  "Outputs": {
    "Index": {
      "Value": {
        "Fn::Index": [
          { "Ref": "ZoneId" },
          { "Ref": "ZoneIds" }
        ]
      }
    }
  }
}
```

Fn::If

如果指定的条件计算为true，则返回一个值；如果指定的条件计算为false，则返回另一个值。在模板Resources和Outputs属性值中支持Fn::If内部函数。您可以使用`ALIYUN::NoValue`伪参数作为返回值来删除相应的属性。

- 声明

```
{"Fn::If": ["condition_name", "value_if_true", "value_if_false"]}
```

- 参数
  - `condition_name`：Conditions中条件对应的条件名称。通过条件名称引用条件。
  - `value_if_true`：当指定的条件计算为true时，返回此值。
  - `value_if_false`：当指定的条件计算为false时，返回此值。
- 示例
  根据输入的参数，确定是否创建数据盘。

```
{
    "ROSTemplateFormatVersion":"2015-09-01",
    "Parameters":{
        "EnvType":{
            "Default":"pre",
            "Type":"String"
        }
    },
    "Conditions":{
        "CreateDisk":{
            "Fn::Equals":[
                "prod",
                {
                    "Ref":"EnvType"
                }
            ]
        }
    },
    "Resources":{
        "WebServer":{
            "Type":"ALIYUN::ECS::Instance",
            "Properties":{
                "DiskMappings":{
                    "Fn::If":[
                        "CreateDisk",
                        [
                            {
                                "Category":"cloud_efficiency",
                                "DiskName":"FirstDataDiskName",
                                "Size":40
                            },
                            {
                                "Category":"cloud_ssd",
                                "DiskName":"SecondDataDiskName",
                                "Size":40
                            }
                        ],
                        {
                            "Ref":"ALIYUN::NoValue"
                        }
                    ]
                },
                "VpcId":"vpc-2zew9pxh2yirtzqxd****",
                "SystemDiskCategory":"cloud_efficiency",
                "SecurityGroupId":"sg-2zece6wcqriejf1v****",
```

```
                "SystemDiskSize":40,
                "ImageId":"centos_6_8_64_40G_base_20170222****",
                "IoOptimized":"optimized",
                "VSwitchId":"vsw-2zed9txvy7h2srqo6****",
                "InstanceType":"ecs.n1.medium"
            }
        }
    },
    "Outputs":{
        "InstanceId":{
            "Value":{
                "Fn::GetAtt":[
                    "WebServer",
                    "InstanceId"
                ]
            }
        },
        "ZoneId":{
            "Value":{
                "Fn::GetAtt":[
                    "WebServer",
                    "ZoneId"
                ]
            }
        }
    }
}
```

- 支持的函数
  - Fn::Or
  - Fn::Not
  - Fn::Equals
  - Fn::FindInMap
  - Fn::And
  - Ref

Fn::Length

内部函数Fn::Length返回对象的长度。

- 声明

```
{"Fn::Length" : Object}
```

- 参数
  Object：需要计算长度的对象。支持字符串、列表和字典三种类型。
- 返回值
  该对象的长度，整型。
- 示例

```
{"Fn::Length": {"key1": "v1", "key2": "v2"}}
```

返回：`2`

```
{"Fn::Length": [1, 2, 3]}
```

返回：`3`

```
{"Fn::Length": "length"}
```

返回：`6`

Fn::ListMerge

合并多个列表为一个列表。

- 声明

```
{"Fn::ListMerge": [["list_1_item_1", "list_1_imte_2", ...], ["list_2_item_1", "list_2_imte_2", ...]]}
```

- 参数
  - `["list_1_item_1", "list_1_imte_2", ...]`：将要合并的第一个列表。
  - `["list_2_item_1", "list_2_imte_2", ...]`：将要和第一个列表合并的列表。
- 示例
  把两个ECS组挂载到同一个负载均衡实例上。

```
{
  "ROSTemplateFormatVersion" : "2015-09-01",
  "Resources" : {
    "LoadBalancer": {
      "Type": "ALIYUN::SLB::LoadBalancer",
      "Properties": {
        "LoadBalancerName": "ros",
        "AddressType": "internet",
        "InternetChargeType": "paybybandwidth",
      }
    },
    "BackendServer1": {
      "Type": "ALIYUN::ECS::InstanceGroup",
      "Properties": {
        "ImageId" : "m-2ze9uqi7wo61hwep****",
        "InstanceType": "ecs.t1.small",
        "SecurityGroupId": "sg-2ze8yxgempcdsq3i****",
        "MaxAmount": 1,
        "MinAmount": 1
      }
    },
    "BackendServer2": {
      "Type": "ALIYUN::ECS::InstanceGroup",
      "Properties": {
        "ImageId" : "m-2ze9uqi7wo61hwep****",
        "InstanceType": "ecs.t1.small",
        "SecurityGroupId": "sg-2ze8yxgempcdsq3i****",
        "MaxAmount": 1,
        "MinAmount": 1
      }
    },
    "Attachment": {
      "Type": "ALIYUN::SLB::BackendServerAttachment",
      "Properties": {
        "LoadBalancerId": {"Ref": "LoadBalancer"},
        "BackendServerList": { "Fn::ListMerge": [
            {"Fn::GetAtt": ["BackendServer1", "InstanceIds"]},
            {"Fn::GetAtt": ["BackendServer2", "InstanceIds"]}
          ]
        }
      }
    }
  }
}
```

- 支持的函数
  - Fn::Base64Encode
  - Fn::GetAtt
  - Fn::Join
  - Fn::Select
  - Ref
  - Fn::If

Fn::GetJsonValue

解析JSON字符串，获取指定的Key在第一层所对应的值。

- 声明

```
{"Fn::GetJsonValue": ["key", "json_string"]}
```

- 参数
  - `key`：键值。
  - `json_string`：指定的需要解析的JSON字符串。

- 示例

WebServer2从WebServer实例执行完UserDat a返回的JSON字符串中，获取对应的值。

```
{
  "ROSTemplateFormatVersion" : "2015-09-01",
  "Resources" : {
    "WebServer": {
      "Type": "ALIYUN::ECS::Instance",
      "Properties": {
        "ImageId" : "m-2ze45uwova5fedlu****",
        "InstanceType": "ecs.n1.medium",
        "SecurityGroupId": "sg-2ze7pxymaix640qr****",
        "Password": "Wenqiao****",
        "IoOptimized": "optimized",
        "VSwitchId": "vsw-2zei67xd9nhcqxzec****",
        "VpcId": "vpc-2zevx9ios1rszqv0a****",
        "SystemDiskCategory": "cloud_ssd",
        "UserData": {"Fn::Join": ["", [
             "#!/bin/sh\n",
             "mkdir ~/test_ros\n",
             "print hello > ~/1.txt\n",
             "Fn::GetAtt": ["WaitConHandle", "CurlCli"],
             "\n",
             "Fn::GetAtt": ["WaitConHandle", "CurlCli"],
             " -d '{\\"id\\" : \\"1\\", \\"data\\": [\\"1111\\", \\"2222\\"]}'\n"
         ]]},
         "PrivateIpAddress": "192.168.XX.XX",
         "HostName": "userdata-1"
      }
    },
    "WaitConHandle": {
        "Type": "ALIYUN::ROS::WaitConditionHandle"
    },
    "WaitCondition": {
        "Type": "ALIYUN::ROS::WaitCondition",
        "Properties": {
          "Handle": {"Ref": "WaitConHandle"},
          "Timeout": 900
        }
    },
    "WebServer2": {
      "Type": "ALIYUN::ECS::Instance",
      "Properties": {
```

```
          "ImageId" : "m-2ze45uwova5fedlu****",
          "InstanceType": "ecs.n1.medium",
          "SecurityGroupId": "sg-2ze7pxymaix640qr****",
          "Password": "Wenqiao****",
          "IoOptimized": "optimized",
          "VSwitchId": "vsw-2zei67xd9nhcqxzec****",
          "VpcId": "vpc-2zevx9ios1rszqv0a****",
          "SystemDiskCategory": "cloud_ssd",
          "UserData":
            {"Fn::Join": ["", [
               "#!/bin/sh\n",
               "mkdir ~/test_ros\n",
               "echo hello > ~/1.txt\n",
               "server_1_token=",
               {"Fn::GetJsonValue": ["1", { "Fn::GetAtt": ["WaitCondition", "Data"]}]},
               "\n"
           ]]},
          "PrivateIpAddress": "192.168.XX.XX",
          "HostName": "userdata-2"
       }
     },
   },
   "Outputs": {
     "InstanceId": {
          "Value" : {"Fn::GetAtt": ["WebServer","InstanceId"]}
     },
     "PublicIp": {
          "Value" : {"Fn::GetAtt": ["WebServer","PublicIp"]}
     }
   }
}
```

- 支持的函数
  - Fn::Base64Encode
  - Fn::GetAtt
  - Fn::Join
  - Fn::Select
  - Ref
  - Fn::If

Fn::MergeMapToList

将多个Mapping合并成一个以Mapping为元素的列表。

- 声明

```
{"Fn::MergeMapToList": [{"key_1": ["key_1_item_1", "key_1_item_2", ...]}, {"key_2":["key_
2_item_1", "key_2_item_2", ...]}, ... ]}
```

- 参数

- `{"key_1": ["key_1_item_1", "key_1_item_2", ...]}`：将要合并的第一个Mapping。`"key_1"`所对应的值必须是一个列表。`"key_1"`将是合并后的列表元素中，每个Mapping的一个键。其对应的值在第一个Mapping中将是`"key_1_item_1"`，在第二个Mapping中是`"key_1_item_2"`，以此类推。最终合并的列表长度是所有将要合并的参数Mapping中，`"key_x"`所对应的列表中最长的长度。如果有的`"key_y"`所对应的列表长度比较短，会重复此列表的最后一个元素，使列表长度都达到最长。
- `{"key_2": ["key_2_item_1", "key_2_item_2", ...]}`：将要合并的第二个Mapping。`"key_2"`所对应的值必须是一个列表。`"key_2"`将是合并后的列表元素中每个Mapping的一个键。其对应的值在第一个Mapping中将是`"key_2_item_1"`，在第二个Mapping中是`"key_2_item_2"`，以此类推。

- 示例
  - 合并三个Mapping，每个Mapping中键值对应的列表长度一致：

```
{
   "Fn::MergeMapToList": [
      {"key_1": ["kye_1_item_1", "kye_1_item_2"]},
      {"key_2": ["kye_2_item_1", "kye_2_item_2"]},
      {"key_3": ["kye_3_item_1", "kye_3_item_2"]}
   ]
}
```

  合并结果如下：

```
[
   {
      "key_1": "kye_1_item_1",
      "key_2": "kye_2_item_1",
      "key_3": "kye_3_item_1"
    },
    {
      "key_1": "kye_1_item_2",
      "key_2": "kye_2_item_2",
      "key_3": "kye_3_item_2"
    }
]
```

- 合并三个Mapping，每个Mapping中键值对应的列表长度不一致：

```
{
   "Fn::MergeMapToList": [
      {"key_1": ["kye_1_item_1", "kye_1_item_2"]},
      {"key_2": ["kye_2_item_1", "kye_2_item_2", "key_2_item_3"]},
      {"key_3": ["kye_3_item_1", "kye_3_item_2"]}
   ]
}
```

合并结果如下：

```
[
   {
      "key_1": "kye_1_item_1",
      "key_2": "kye_2_item_1",
      "key_3": "kye_3_item_1"
    },
    {
      "key_1": "kye_1_item_2",
      "key_2": "kye_2_item_2",
      "key_3": "kye_3_item_2"
    },
    {
      "key_1": "kye_1_item_2",
      "key_2": "kye_2_item_3",
      "key_3": "kye_3_item_2"
    }
]
```

- 以下模板示例中，把WebServer中创建的所有实例，都加入到一个负载均衡的虚拟服务器组中。

```
{
    "ROSTemplateFormatVersion": "2015-09-01",
    "Resources": {
        "WebServer": {
          "Type": "ALIYUN::ECS::InstanceGroupClone",
          "Properties": {
              "SourceInstanceId": "i-xxxxx",
               "Password": "Hello****",
               "MinAmount": 1,
               "MaxAmount": 1
          }
        },
        "CreateVServerGroup": {
            "Type": "ALIYUN::SLB::VServerGroup",
            "Properties": {
                "LoadBalancerId": "lb-****",
                "VServerGroupName": "VServerGroup-****",
                "BackendServers": {
                   "Fn::MergeMapToList": [
                       {"Port": [6666, 9090, 8080]},
                       {"ServerId": {"Fn::GetAtt": ["WebServer", "InstanceIds"]}},
                       {"Weight": [20, 100]}
                   ]
                }
            }
        }
    }
}
```

- 支持的函数
  - Fn::Base64Encode
  - Fn::GetAtt
  - Fn::Join
  - Fn::Select
  - Ref
  - Fn::If
  - Fn::ListMerge
  - Fn::GetJsonValue

Fn::Avg

内部函数Fn::Avg对一组数求平均值。

- 声明

```
{"Fn::Avg": [ndigits, [number1, number2, ...]]}
```

- 参数
  - `ndigits`：平均值保留几位小数，必须为整数。

- `[number1, number2, ...]`：一组用来求取平均值的数字。组内每个元素可以是数字或能转换成数字的字符串。

- 返回值
  给定一组数字的平均值。
- 示例

```
{"Fn::Avg": [ 1, [1, 2, 6.0]]}
{"Fn::Avg": [ 1, ['1', '2', '6.0']]}
```

  返回：3.0。
- 支持的函数
  - Fn::GetAtt
  - Ref

Fn::SelectMapList

内部函数Fn::SelectMapList返回一个由map中元素构成的列表。

- 声明

```
{"Fn::SelectMapList": ["key2", [{"key1": "value1-1", "key3": "value1-3"}, {"key1": "value
2-1", "key2": "value2-2"}, {"key1": "value3-1", "key2": "value3-2"}, ...] ] }
```

- 参数
  - `key2`：在map中查询的key。
  - `[{ "key1": "value1-1", "key3": "value1-3" }, ... ]`：由map组成的List。
- 返回值
  对map_list中的每个map，取出key对应的值，合并成一个列表。
- 示例

```
{
 "Fn::SelectMapList": [
   "key2",
   [
      {"key1": "value1-1", "key3": "value1-3"},
      {"key1": "value2-1", "key2": "value2-2"},
      {"key1": "value3-1", "key2": "value3-2"}
   ]
 ]
}
```

  返回：`["value2-2","value3-2"]`。

Fn::Add

内部函数Fn::Add对参数进行求和。

- 声明

```
{"Fn::Add": [{"Product": "ROS"}, {"Fn": "Add"}]}
```

- 参数
  - 参数为一个列表。

- 列表内元素可以是数字、列表、字典，但是所有元素必须是同一类型，至少2个元素。

- 返回值
  如果是数字类型，对参数进行求和；如果是列表类型，对参数进行拼接；如果是字典类型，对参数进行合并，key相同的情况下，后面的覆盖前面的。
- 示例

```
{
 "Fn::Add": [
   {"Product": "ROS"},
   {"Fn": "Add"}
 ]
}
```

返回：`{"Fn":"Add","Product":"ROS"}`。

Fn::Calculate

内部函数Fn::Calculate对字符串形式的表达式进行计算。

- 声明

```
{"Fn::Calculate" : [expression, ndigits, [<number1>, <number2>, ... ]]}
```

- 参数
  - `expression`：字符串形式的表达式。
  - `ndigits`：取值：0或正整数，表示保留小数的位数，如果表达式中不包含浮点数，则此参数不生效。
  - `[<number0>, <number1>, <number2>, ... ]`：非必需参数。expression中可以定义{n}，n为列表中某个number的索引，在计算表达式时用number的值替换{n}。

- 返回值
  表达式的计算结果，类型为Number。
- 示例

```
{"Fn::Calculate": ["(2+3)/2*3-1", 1]}
{"Fn::Calculate": ["(2.0+3)/2*3-1", 1]}
{"Fn::Calculate": ["({1}+3)/2*3-1", 1, [3, 5, 6]]}
{"Fn::Calculate": ["({0}+{1})%3", 0, [5, 6]]}
```

返回

```
5
6.5
11
2
```

> **说明** 使用整数进行除法运算时不保留小数，即5/2=2，因此`{"Fn::Calculate": ["(2+3)/2*3-1", 1]}`返回值为`5`。

Fn::Max

内部函数Fn::Max用于获取由数字组成的列表中的最大数字。

- 声明

```
{"Fn::Max": [Number1, Number2, Number3, ...]}
```

- 返回值
  列表中的最大数字。
- 示例

```
{"Fn::Max": [1.1, 2, 3]}
```

  返回

```
3
```

Fn::Min

内部函数Fn::Min用于获取由数字组成的列表中的最小数字。

- 声明

```
{"Fn::Min": [Number1, Number2, Number3, ...]}
```

- 返回值
  列表中的最小数字。
- 示例

```
{"Fn::Min": [1.1, 2, 3]}
```

  返回

```
1.1
```

Fn::GetStackOutput

内部函数Fn::GetStackOutput用于获取指定资源栈的输出。

- 声明

```
{"Fn::GetStackOutput": [Stack, OutputName, RegionId]}
```

- 参数
  - `Stack`：必选，String类型。资源栈的名称或ID。
  - `OutputName`：必选，String类型。资源栈某一输出名。
  - `RegionId`：可选，String类型。资源栈所属地域，若不指定，则与调用资源栈相同。
- 返回值
  资源栈Stack的名为OutputName的输出，类型依赖于输出值，不确定。若符合如下条件则返回空值：
  - 传递的参数中有一个值为空值。
  - Stack不存在（含已删除）。
  - Stack运行中或删除失败。
  - Output不存在。
- 限制
  - 单向引用：A->B->C（允许），A->B->A（不允许），A->A（不允许）。
  - 引用最大长度为3：A->B->C（允许），A->B->C->D（不允许）。

- 示例

```
{"Fn::GetStackOutput": ["4a6c9851-3b0f-4f5f-b4ca-a14bf691****", "InstanceId"]}
```

Fn::Jq

内部函数Fn::Jq用于支持Jq功能。Jq功能详情请参见Jq文档。

- 声明

```
{"Fn::Jq": [method,script,object]}
```

- 参数
  - `method`：必选，字符串。取值：
    - First：符合条件的第一个值。
    - All：符合条件的所有值。
  - `script`：必选，字符串。Jq脚本。
  - `object`：必选，JSON字符串。
- 返回值
  选定的字符串。
- 示例
  - 使用过滤器 `.test` 过滤出的值取第一个。当给定一个JSON字符串作为输入时，它将在键 `test` 处产生值；如果值不存在，则为null。

    ```
    {"Fn::Jq":["First",".test","{\"test\":\"test\"}"]}
    ```

    返回

    ```
    "test"
    ```

- 使用过滤器 `.parameters[]` 过滤出的内容，再次通过新的过滤器进行过滤。它们之间通过竖线（|）进行连接。给定一个JSON字符串作为输入时，它将在键 `.parameters[] | {\"param_name\": .name, \"param_type\":.type}` 处产生值；如果值不存在，则为null。

```
{
    "Fn::Jq": [
        "All",
        ".parameters[] | {\"param_name\": .name, \"param_type\":.type}",
        {
            "changeSet": {
                "items": [
                ],
                "kind": "git"
            },
            "id": "2013-12-27_00-09-37",
            "parameters": [
                {
                    "name": "PKG_TAG_NAME",
                    "value": "trunk"
                },
                {
                    "name": "GIT_COMMIT",
                    "value": "master"
                },
                {
                    "name": "TRIGGERED_JOB",
                    "value": "trunk-buildall"
                }]
        }]
}
```

返回

```
"[{u'param_name': u'PKG_TAG_NAME', u'param_type': None}, {u'param_name': u'GIT_COMMIT', u'param_type': None}, {u'param_name': u'TRIGGERED_JOB', u'param_type': None}]"
```

Fn::FormatTime

内部函数Fn::FormatTime用于返回格式化后的当前时间。

- 声明
  - 根据指定时区，获取该时区的当前时间。

```
{"Fn::FormatTime": ["format","<time_zone>"]}
```

  - 获取默认UTC时区的当前时间。

```
{"Fn::FormatTime": "format"}
```

- 参数
  - `format`：必选，日期格式字符串，例如：`"%Y-%m-%d %H:%M:%S"`。
    日期格式字符串中字段含义如下：

字段	说明	示例
%y	补零后，以十进制数表示不带世纪的年份。例如：2001年表示为01，0相当于占位符。	21
%Y	以十进制数表示带世纪的年份。	2021
%m	补零后，以十进制数表示的月份。	04
%d	补零后，以十进制数表示的一个月中的一天。	07
%H	补零后，以十进制数表示的小时（24小时制）。	14
%I	补零后，以十进制数表示的小时（12小时制）。	08
%M	补零后，以十进制数表示的分钟。	09
%S	补零后，以十进制数表示的秒。	10
%a	星期的缩写。	Wed
%A	星期的全称。	Wednesday
%b	月份的缩写。	Apr
%B	月份的全称。	April
%c	日期和时间。	Wed Apr 7 08:15:10 2021
%j	补零后，以十进制数表示的一年中的一天。	097
%p	取值： ■ AM：上午。 ■ PM：下午。	AM
%U	补零后，以十进制数表示的一年中的周序号（星期日作为每周的第一天）。 **说明** 在新的一年中第一个星期日之前的所有日子都被视为第0周。	14
%w	以十进制数显示的星期中的一天。 0表示星期日，6表示星期六。	3
%W	以十进制数表示的一年中的周序号（星期一作为每周的第一天）。 **说明** 在新的一年中第一个星期一之前的所有日子都被视为第0周。	14

字段	说明	示例
%x	日期。	04/07/21
%X	时间。	08:15:10
%Z	时区名称。取值： ■ UTC ■ GMT	UTC
%f	补零后，以十进制数表示微秒。	091798
%z	UTC偏移量，格式为 `+HHMM` 或 `-HHMM` 。	+0000
%%	百分号（%）。	%

  - `time_zone` ：时区，例如： `UTC(default), Asia/Shanghai` 。

- 返回值
  当前时间。
- 示例

```
{"Fn::FormatTime": "%Y-%m-%d %H:%M:%S"}
```

返回

```
"2021-06-11 03:48:19"
```

```
{"Fn::FormatTime": ["%Y-%m-%d %H:%M:%S","Asia/Shanghai"]}
```

返回

```
"2021-06-11 12:01:25"
```

Fn::MarketplaceImage

内部函数Fn::MarketplaceImage用于返回指定云市场镜像商品Code的默认镜像ID。

> **说明** 在更新资源栈或创建更改集时，若此函数返回值和前一次不同，会被认为有差异而触发引用此函数的资源更新。例如：假设某镜像商品Code在杭州地域 `cn-hangzhou` 查到的镜像ID是 `m-1` ，您使用包含此函数的模板创建了资源栈。如果后续云市场商家将 `m-1` 更换为 `m-2` ，您再次使用包含此函数的模板更新资源栈时，由于得到的镜像ID不同而被认为有差异，从而触发引用此镜像ID的资源（例如ALIYUN::ECS::InstanceGroup）的更新。

- 声明

```
{"Fn::MarketplaceImage" : "<ImageProductCode>" }
```

- 参数
  `ImageProductCode` ：必选，字符串。云市场镜像商品Code。例如： `cmjj02****` 。
- 返回值
  字符串。云市场镜像商品Code在当前地域对应的默认镜像ID。

- 示例

```
{"Fn::MarketplaceImage": 'cmjj02****'}
```

返回：`m-2zeeyr9nxxoo5gcg****`。

Fn::Any

内部函数Fn::Any用于返回指定数组中取值的真假情况。当数组中任意一项为真时，返回true（真），否则返回false（假）。

- 声明

```
{ "Fn::Any": [ value_1, value_2, ... ] }
```

- 参数
  `value`：必选，任意类型的值。例如：true、false、1、0等。
- 返回值
  true或false。
- 示例
  - 示例1

    ```
    {"Fn::Any": [true, true]}
    ```

    返回：`true`。
  - 示例2

    ```
    {"Fn::Any": [true, false]}
    ```

    返回：`true`。
  - 示例3

    ```
    {"Fn::Any": [false, false]}
    ```

    返回：`false`。
  - 示例4

    ```
    {"Fn::Any": [1, 1]}
    ```

    返回：`true`。
  - 示例5

    ```
    {"Fn::Any": [0, 0]}
    ```

    返回：`false`。

6.2.6. 映射（Mappings）

映射是一个Key-Value映射表。在模板的Resources和Outputs中，可以使用Fn::FindInMap内部函数，通过指定Key而获取映射表的Value。

语法

映射由Key-Value对组成。其中Key和Value可以为字符串类型或者数字类型。如果声明多个映射，用英文逗号（,）分隔。每个映射的名称不能重复。

> **说明** 映射须为纯数据，映射中不能使用函数。

示例

- 正确的映射示例

```
"Mappings": {
    "ValidMapping": {
        "TestKey1": {"TestValu1": "value1"},
        "TestKey2": {"TestValu2": "value2"},
        1234567890: {"TestValu3": "value3"},
        "TestKey4": {"TestValu4": 1234}
    }
}
```

- 错误的映射示例

```
"Mappings": {
    "InvalidMapping1": {
        "ValueList": ["foo", "bar"],
          "ValueString": "baz"
    },
    "InvalidMapping2": ["foo", {"bar" : "baz"}],
    "InvalidMapping3": "foobar"
}
```

- 使用内部函数Fn::FindInMap返回对应的值示例

```
{
  "ROSTemplateFormatVersion": "2015-09-01",
  "Parameters": {
    "regionParam": {
      "Description": "选择创建ECS的地域",
      "Type": "String",
      "AllowedValues": [
        "hangzhou",
        "beijing"
      ]
    }
  },
  "Mappings": {
    "RegionMap": {
      "hangzhou": {
        "32": "m-25l0rcfjo",
        "64": "m-25l0rcfj1"
      },
      "beijing": {
        "32": "m-25l0rcfj2",
        "64": "m-25l0rcfj3"
      }
    }
  },
  "Resources": {
    "WebServer": {
```

```
      "Type": "ALIYUN::ECS::Instance",
      "Properties": {
        "ImageId": {
          "Fn::FindInMap": [
            "RegionMap",
            {
              "Ref": "regionParam"
            },
            "32"
          ]
        },
        "InstanceType": "ecs.t1.small",
        "SecurityGroupId": "sg-25zwc****",
        "ZoneId": "cn-beijing-b",
        "Tags": [
          {
            "Key": "Department1",
            "Value": "HumanResource"
          },
          {
            "Key": "Department2",
            "Value": "Finance"
          }
        ]
      }
    }
  }
}
```

6.2.7. 条件（Conditions）

条件由Fn::And、Fn::Or、Fn::Not或Fn::Equals定义，根据您在创建或更新资源栈时，指定的输入参数值进行计算。在每个条件中，都可以引用其他条件、参数值或映射。

语法

每个条件由条件名和条件本身组成，其中条件名是字符串类型。多个条件用半角逗号（,）隔开，每个条件名不能重复。

模板的Resources和Outputs部分，将条件与资源关联起来。条件和资源的关联通过两种方式实现：Fn::If函数或者资源的Condition字段。

允许使用如下函数，但不能是最外层的函数：

Fn::Select、 Fn::Join、 Fn::Split、Fn::Replace、Fn::Base64Encode、Fn::Base64Decode、Fn::MemberListToMap、Fn::If、 Fn::ListMerge、 Fn::GetJsonValue、Fn::MergeMapToList、Fn::SelectMapList、Fn::Add、Fn::Avg、Fn::Str、Fn::Calculate、Ref（仅参数引用）、Fn:: FindInMap、Fn::Max、Fn::Min。

示例

- 设置Conditions

```
"Conditions" : {
       "DevEnv": {"Fn::Equals": ["Dev", {"Ref": "EnvType"}]},
       "UTEnv": {"Fn::Equals": ["UT", {"Ref": "EnvType"}]},
       "PREEnv": {"Fn::Not": {"Fn::Or": ["DevEnv", "UTEnv"]}},
       "ProdEnv": {"Fn::And": [{"Fn::Equals": ["Prod", {"Ref": "EnvType"}]}, "PREEnv"]}
}
```

- 关联Conditions和Resources
  本示例中，根据EnvType参数值决定是否为ECS instance创建数据盘和OSS bucket。

```
{
    "ROSTemplateFormatVersion":"2015-09-01",
    "Parameters":{
        "EnvType":{
            "Default":"pre",
            "Type":"String"
        }
    },
    "Conditions":{
        "CreateProdRes":{
            "Fn::Equals":[
                "prod",
                {
                    "Ref":"EnvType"
                }
            ]
        }
    },
    "Resources":{
        "WebServer":{
            "Type":"ALIYUN::ECS::Instance",
            "Properties":{
                "DiskMappings":{
                    "Fn::If":[
                        "CreateProdRes",
                        [
                            {
                                "Category":"cloud_efficiency",
                                "DiskName":"FirstDataDiskName",
                                "Size":40
                            },
                            {
                                "Category":"cloud_ssd",
                                "DiskName":"SecondDataDiskName",
                                "Size":40
                            }
                        ],
                        {
                            "Ref":"ALIYUN::NoValue"
                        }
                    ]
                },
                "VpcId":"vpc-2zew9pxh2yirtzqxd****",
                "SystemDiskCategory":"cloud_efficiency",
```

```
                "SecurityGroupId":"sg-2zece6wcqriejf1v****",
                "SystemDiskSize":40,
                "ImageId":"centos_6_8_64_40G_base_2017****.vhd",
                "IoOptimized":"optimized",
                "VSwitchId":"vsw-2zed9txvy7h2srqo6****",
                "InstanceType":"ecs.n1.medium"
            }
        },
        "OssBucket": {
            "Type": "ALIYUN::OSS::Bucket",
            "Condition": "CreateProdRes",
            "Properties": {
                "AccessControl": "private",
                "BucketName": "myprodbucket"
            }
        }
    },
    "Outputs":{
        "InstanceId":{
            "Value":{
                "Fn::GetAtt":[
                    "WebServer",
                    "InstanceId"
                ]
            }
        },
        "OssDomain":{
            "Condition": "CreateProdRes",
            "Value":{
                "Fn::GetAtt":[
                    "OssBucket",
                    "DomainName"
                ]
            }
        }
    }
}
```

6.2.8. 伪参数（Pseudo parameters）

伪参数是资源编排服务ROS（Resource Orchestration Service）的编排引擎提供的固定参数。它们可以和用户定义的参数一样被引用，其值在ROS运行时确定。
ROS提供了以下伪参数：

- `ALIYUN::StackName`：资源栈名称。
- `ALIYUN::StackId`：资源栈ID。
- `ALIYUN::Region`：资源栈所在地域。
- `ALIYUN::AccountId`：资源栈账户ID。
- `ALIYUN::TenantId`：当前账户的阿里云账号ID。
- `ALIYUN::NoValue`：创建或更新资源栈时，如果`ALIYUN::NoValue`用于可选属性，则将删除该属性；如果`ALIYUN::NoValue`用于必选属性，则将按类型获取默认值（例如，用于String类型的属性值为空字符串；用于Integer类型的属性值为0；用于数组类型属性值为空数组等）。

- `ALIYUN::Index`：一个特殊的伪参数，仅在资源`Count`功能中使用，其他情况不能使用。`Count`详情，请参见Count。

示例

`JSON`格式

```
{
  "ROSTemplateFormatVersion": "2015-09-01",
  "Parameters": {
    "EnvType": {
      "Default": "pre",
      "Type": "String"
    }
  },
  "Conditions": {
    "CreateDisk": {
      "Fn::Equals": [
        "prod",
        {
          "Ref": "EnvType"
        }
      ]
    }
  },
  "Resources": {
    "WebServer": {
      "Type": "ALIYUN::ECS::Instance",
      "Properties": {
        "DiskMappings": {
          "Fn::If": [
            "CreateDisk",
            [
              {
                "Category": "cloud_efficiency",
                "DiskName": "FirstDataDiskName",
                "Size": 40
              },
              {
                "Category": "cloud_ssd",
                "DiskName": "SecondDataDiskName",
                "Size": 40
              }
            ],
            {
              "Ref": "ALIYUN::NoValue"
            }
          ]
        },
        "VpcId": "vpc-m5eebunc50zfbmts7****",
        "SystemDiskCategory": "cloud_efficiency",
        "SecurityGroupId": "sg-m5eagh7rzys2z8sa****",
        "SystemDiskSize": 40,
        "ImageId": "cent****",
        "IoOptimized": "optimized",
```

```
        "VSwitchId": "vsw-m5eem62p9729y6gps****",
        "InstanceType": "ecs.c5.large"
      }
    }
  },
  "Outputs": {
    "StackName": {
      "Value": {
        "Ref": "ALIYUN::StackName"
      }
    },
    "StackId": {
      "Value": {
        "Ref": "ALIYUN::StackId"
      }
    },
    "Region": {
      "Value": {
        "Ref": "ALIYUN::Region"
      }
    },
    "UserID": {
      "Value": {
        "Ref": "ALIYUN::AccountId"
      }
    }
  }
}
```

`YAML` 格式

```
ROSTemplateFormatVersion: '2015-09-01'
Parameters:
  EnvType:
    Default: pre
    Type: String
Conditions:
  CreateDisk:
    Fn::Equals:
    - prod
    - Ref: EnvType
Resources:
  WebServer:
    Type: ALIYUN::ECS::Instance
    Properties:
      DiskMappings:
        Fn::If:
        - CreateDisk
        - - Category: cloud_efficiency
            DiskName: FirstDataDiskName
            Size: 40
          - Category: cloud_ssd
            DiskName: SecondDataDiskName
            Size: 40
        - Ref: ALIYUN::NoValue
      VpcId: vpc-m5eebunc50zfbmts7****
      SystemDiskCategory: cloud_efficiency
      SecurityGroupId: sg-m5eagh7rzys2z8sa****
      SystemDiskSize: 40
      ImageId: cent****
      IoOptimized: optimized
      VSwitchId: vsw-m5eem62p9729y6gps****
      InstanceType: ecs.c5.large
Outputs:
  StackName:
    Value:
      Ref: ALIYUN::StackName
  StackId:
    Value:
      Ref: ALIYUN::StackId
  Region:
    Value:
      Ref: ALIYUN::Region
  UserID:
    Value:
      Ref: ALIYUN::AccountId
```

6.2.9. 元数据（Metadata）

元数据（Metadata）用于对Parameters中定义的参数进行分组，并且可以为每一组分别定义标签。
在资源编排控制台上创建资源栈时，Parameters中定义的参数会展示在**配置模板参数**的**参数录入**中。
Metadata用于对Parameters中定义的参数进行分组，并且可以为每一组分别定义标签。

语法

```
"Metadata": {
    "ALIYUN::ROS::Interface": {
        "ParameterGroups": [
            //ParameterGroups为必填项
            <Group1>,
            <Group2>,
            ...
        ],
        "TemplateTags":  [
            //TemplateTags可选，为自定义字符串
            <Tag1>,
            <Tag2>,
        ]
    }
}
```

Group语法

```
{
    "Parameters": [
        //Parameters为必填项
        <Parameter1>,
        <Parameter2>,
        ...
    ],
    "Label": {
        //Label为必填项
        "default": <自定义字符串>
    }
}
```

示例

```
{
  "ROSTemplateFormatVersion": "2015-09-01",
  "Description": "Metadata演示",
  "Metadata": {
    "ALIYUN::ROS::Interface": {
      "ParameterGroups": [
        {
          "Parameters": [
            "VpcName",
            "VpcCidrBlock"
          ],
          "Label": {
            "default": "VPC"
          }
        },
        {
          "Parameters": [
            "VswName",
            "VswCidrBlock"
```

```
          ],
          "Label": {
            "default": "VSwitch"
          }
        }
      ],
      "TemplateTags": [
        "VPC-VSwitch"
      ]
    }
  },
  "Parameters": {
    "VpcName": {
      "Type": "String",
      "Label": "名称",
      "Description": "长度为2~128个字符，以英文字母或汉字开头，可包含英文字母、汉字、数字、下划线（_）和短划线（-）。",
      "Default": "MyVPC"
    },
    "VswName": {
      "Type": "String",
      "Label": "名称",
      "Description": "长度为2~128个字符，以英文字母或汉字开头，可包含英文字母、汉字、数字、下划线（_）和短划线（-）。",
      "Default": "MyVSwitch"
    },
    "VpcCidrBlock": {
      "Type": "String",
      "AllowedValues": [
        "10.0.0.0/8",
        "172.16.0.0/12",
        "192.168.0.0/16"
      ],
      "Description": "专有网络网段。如需自定义网段，请更改模板此参数的可用值，网段必须属于可用值范围的子网网段。",
      "Label": "IPv4网段",
      "Default": "192.168.0.0/16"
    },
    "VswCidrBlock": {
      "Type": "String",
      "Description": "必须是所属专有网络的子网段，并且没有被其他交换机占用。",
      "Label": "IPv4网段",
      "Default": "192.168.1.0/24"
    }
  },
  "Resources": {
    "VSwitch": {
      "Type": "ALIYUN::ECS::VSwitch",
      "Properties": {
        "VpcId": {
          "Ref": "VPC"
        },
        "ZoneId": {
          "Fn::Select": [
```

```
            "1",
            {
              "Fn::GetAZs": {
                "Ref": "ALIYUN::Region"
              }
            }
          ]
        },
        "CidrBlock": {
          "Ref": "VswCidrBlock"
        },
        "VSwitchName": {
          "Ref": "VswName"
        }
      }
    },
    "VPC": {
      "Type": "ALIYUN::ECS::VPC",
      "Properties": {
        "CidrBlock": {
          "Ref": "VpcCidrBlock"
        },
        "VpcName": {
          "Ref": "VpcName"
        }
      }
    }
  }
}
```

6.2.10. 转换（Transform）

转换用于简化特定场景的模板编写复杂度，可通过在模板中指定Transform进行声明。ROS会将含有Transform的模板转换为普通模板，然后进行资源的创建、更新等操作。
ROS支持Aliyun::Serverless转换，可用于基于函数计算服务的无服务器（Serverless）场景。更多信息，请参见无服务器架构。

Aliyun::Serverless转换会将使用SAM（Serverless Application Model）语法编写的模板（基于阿里云Funcraft工具），转换为ROS的普通模板。

更多信息，请参见什么是Serverless Devs和SAM（Serverless Application Model） 语法。

支持的资源类型

- Aliyun::Serverless::Api
- Aliyun::Serverless::CustomDomain
- Aliyun::Serverless::Function
- Aliyun::Serverless::Log
- Aliyun::Serverless::Log::Logstore
- Aliyun::Serverless::MNSTopic
- Aliyun::Serverless::Service
- Aliyun::Serverless::TableStore

- Aliyun::Serverless::TableStore::Table

语法

转换声明的值必须为字符串，不能使用参数或函数来指定转换值。

JSON 示例

```
"Transform" : "Aliyun::Serverless-2018-04-03"
```

YAML 示例

```
Transform: Aliyun::Serverless-2018-04-03
```

示例

以下是一个使用阿里云SAM语法编写的模板：

```
ROSTemplateFormatVersion: "2015-09-01"
Transform: "Aliyun::Serverless-2018-04-03"
Resources:
  MyService: # service name
    Type: "Aliyun::Serverless::Service"
    Properties:
      Policies:
        - AliyunFCReadOnlyAccess # Managed Policy
        - Version: "1" # Policy Document
          Statement:
            - Effect: Allow
              Action:
                - oss:GetObject
                - oss:GetObjectACL
              Resource: "*"
    MyFunction: # function name
      Type: "Aliyun::Serverless::Function"
      Properties:
        Handler: index.handler
        Runtime: nodejs6
        CodeUri: "oss://testBucket/myCode.zip"
```

使用模板创建更改集或者资源栈时，ROS会展开SAM语法。上述模板将会转换以下内容：

- 将Aliyun::Serverless::Service资源转换为ALIYUN::FC::Service资源。
- 将Aliyun::Serverless::Service资源中定义的Policies转换为ALIYUN::RAM::Role和ALIYUN::RAM::AttachPolicyToRole资源，并指定在 ALIYUN::FC::Service的Role参数中。
- 将Aliyun::Serverless::Function资源转换为ALIYUN::FC::Function资源。

转换后的ROS模板如下：

```
{
  "ROSTemplateFormatVersion": "2015-09-01",
  "Resources": {
    "MyServiceRole": {
      "Type": "ALIYUN::RAM::Role",
      "Properties": {
        "RoleName": "test-MyServiceRole-7840BA19B01C",
```

```
        "Description": "Function Compute default role",
        "Policies": [
          {
            "PolicyName": "test-MyServiceRole-7840BA19B01CPolicy1",
            "PolicyDocument": {
              "Version": "1",
              "Statement": [
                {
                  "Action": ["oss:GetObject", "oss:GetObjectACL"],
                  "Resource": ["*"],
                  "Effect": "Allow"
                }
              ]
            }
          }
        ],
        "AssumeRolePolicyDocument": {
          "Version": "1",
          "Statement": [
            {
              "Action": "sts:AssumeRole",
              "Effect": "Allow",
              "Principal": {
                "Service": ["fc.aliyuncs.com"]
              }
            }
          ]
        }
      }
    },
    "AliyunFCReadOnlyAccessMyServiceRole": {
      "Type": "ALIYUN::RAM::AttachPolicyToRole",
      "Properties": {
        "PolicyName": "AliyunFCReadOnlyAccess",
        "PolicyType": "System",
        "RoleName": {
          "Fn::GetAtt": ["MyServiceRole", "RoleName"]
        }
      },
      "DependsOn": "MyServiceRole"
    },
    "MyService": {
      "Type": "ALIYUN::FC::Service",
      "Properties": {
        "ServiceName": "test-MyService-E522E1B83D81",
        "Role": {
          "Fn::GetAtt": ["MyServiceRole", "Arn"]
        },
        "LogConfig": {
          "Project": "",
          "Logstore": ""
        },
        "InternetAccess": true
      },
```

```
      "DependsOn": ["MyServiceRole", "AliyunFCReadOnlyAccessMyServiceRole"]
    },
    "MyServiceMyFunction": {
      "Type": "ALIYUN::FC::Function",
      "Properties": {
        "Code": {
          "OssBucketName": "testBucket",
          "OssObjectName": "myCode.zip"
        },
        "FunctionName": "MyFunction",
        "ServiceName": "test-MyService-E522E1B83D81",
        "EnvironmentVariables": {
          "PATH": "/code/.fun/root/usr/local/bin:/code/.fun/root/usr/local/sbin:/code/.fun/
root/usr/bin:/code/.fun/root/usr/sbin:/code/.fun/root/sbin:/code/.fun/root/bin:/code/.fun/p
ython/bin:/usr/local/bin:/usr/local/sbin:/usr/bin:/usr/sbin:/sbin:/bin",
          "LD_LIBRARY_PATH": "/code/.fun/root/usr/lib:/code/.fun/root/usr/lib/x86_64-linux-
gnu:/code/.fun/root/lib/x86_64-linux-gnu:/code/.fun/root/usr/lib64:/code:/code/lib:/usr/loc
al/lib",
          "PYTHONUSERBASE": "/code/.fun/python"
        },
        "Handler": "index.handler",
        "Runtime": "nodejs6"
      },
      "DependsOn": ["MyService"]
    }
  }
}
```

6.3. 自定义资源

使用自定义资源，您可以在模板中编写自定义配置逻辑，每次创建、更新 (如果您更改了自定义资源) 或删除资源栈时，阿里云 ROS 都会运行该逻辑。例如，您可能需要包含不可作为 阿里云 ROS 资源类型的资源。您可以使用自定义资源包含这些资源。这样，您仍然可以在一个资源栈中管理所有相关资源。

使用 ALIYUN::ROS::CustomResource 或 Custom::MyCustomResourceTypeName 资源类型在模板中定义自定义资源。自定义资源需要一个属性，即服务令牌，它指定 阿里云 ROS 发送请求的目标，如 阿里云 MNS(消息服务) 主题&队列， 阿里云 FC(函数计算) 函数，或 HTTP&HTTPS 服务。

自定义资源必须将响应发送到预签名的响应 URL。如果不能向 ROS 发送响应，阿里云 ROS 不会收到响应，资源栈操作就会失败。ResponseURL提供了公网响应的能力，InnerResponseURL提供了阿里云内网响应的能力。

自定义资源的工作原理

对自定义资源执行的任何操作均涉及三方。

- template developer
  - 创建包含自定义资源类型的模板。template developer在模板中指定服务令牌和所有输入数据。
- custom resource provider
  - 拥有自定义资源并确定如何处理和响应来自 阿里云 ROS 的请求。custom resource provider必须提供template developer使用的服务令牌。
- **阿里云 ROS**

- 在资源栈操作期间，向模板中指定的服务令牌发送请求，然后等待响应，再继续资源栈操作。

template developer和custom resource provider可以是同一人员或实体，但过程相同。以下步骤介绍了一般过程：

1. template developer在其模板中定义自定义资源，该模板包含服务令牌和任何输入数据参数。根据自定义资源，输入数据是可能必需的；但是，服务令牌是总是必需的。
   服务令牌指定 阿里云 ROS 将请求发送到的位置，例如发送到 阿里云 MNS 主题 ARN 或 阿里云 FC 函数 ARN。有关更多信息，请参阅 ALIYUN::ROS::CustomResource。服务令牌和输入数据的结构由custom resource provider定义。
2. 当任何人使用模板创建、更新或删除自定义资源时，阿里云 ROS 将向指定服务令牌发送请求。服务令牌无区域限制。
   在请求中，阿里云 ROS 包含请求类型和自定义资源向其发送请求的预签名 URL 等信息。有关请求中包含的内容的更多信息，请参阅自定义资源请求对象。
   以下示例数据显示 阿里云 ROS 在请求中包含哪些内容：

```
{
   "RequestType" : "Create",
   "RequestId" : "unique id for this create request",
   "ResponseURL" : "pre-signed-url-for-create-response",
   "InnerResponseURL" : "pre-signed-inner-url-for-create-response",
   "ResourceType" : "Custom::MyCustomResourceType",
   "LogicalResourceId" : "name of resource in template",
   "StackId" : "stack id",
   "StackName" : "stack name",
   "ResourceOwnerId": "resource owner id",
   "CallerId": "caller id",
   "RegionId": "region id",
   "ResourceProperties" : {
      "key1" : "string",
      "key2" : [ "list" ],
      "key3" : { "key4" : "map" }
   }
}
```

3. custom resource provider处理 阿里云 ROS 请求并向预签名 URL 返回 SUCCESS 或 FAILED 响应。
   custom resource provider 提供采用 JSON 格式数据响应 URL。
   在响应中，custom resource provider还可以包含template developer可以访问的名称-值对。例如，如果请求成功，响应可以包含输出数据，如果请求失败，可以包含错误消息。有关响应的更多信息，请参阅自定义资源响应对象。
   custom resource provider负责侦听和响应请求。例如，对于 阿里云 MNS主题 通知，custom resource provider 必须侦听并响应发送到特定主题 ARN 的通知。阿里云 ROS 在预签名 URL 位置等待并侦听响应。
   以下示例数据说明自定义资源在响应中可以包含的内容：

```
{
   "Status" : "SUCCESS",
   "RequestId" : "unique id for this create request (copied from request)",
   "LogicalResourceId" : "name of resource in template (copied from request)",
   "StackId" : "stack id (copied from request)",
   "PhysicalResourceId" : "required vendor-defined physical id that is unique for that
vendor",
   "Data" : {
      "keyThatCanBeUsedInGetAtt1" : "data for key 1",
      "keyThatCanBeUsedInGetAtt2" : "data for key 2"
   }
}
```

4. 获得 SUCCESS 响应后，阿里云ROS 继续堆栈操作。如果收到 FAILED 响应或未返回任何响应，则操作失败。来自自定义资源的所有输出数据都由预签名 URL 响应返回。template developer可使用 Fn::GetAtt 函数检索该数据。

6.3.1. 概览

使用自定义资源，您可以在模板中编写自定义配置逻辑，当您创建、更新（如果您更改了自定义资源）或删除资源栈时，ROS会运行该逻辑。例如：您可能需要使用不可作为ROS资源类型的资源，此时可以在模板中自定义资源，从而在一个资源栈中管理所有相关资源。

自定义资源介绍

您可以使用ALIYUN::ROS::CustomResource或Custom::MyCustomResourceTypeName资源类型在模板中定义自定义资源。自定义资源需要一个属性，即服务令牌，它指定ROS发送请求的目标，如MNS（消息服务）主题和队列，FC（函数计算）函数或HTTP和HTTPS服务。

自定义资源必须将响应发送到预签名的响应URL。如果不能向ROS发送响应，资源栈操作会失败。ResponseURL提供了公网响应的能力，IntranetResponseURL提供了阿里云内网响应的能力。

自定义资源的工作原理

自定义资源执行的操作涉及以下对象：

- template developer：创建包含自定义资源类型的模板。template developer在模板中指定服务令牌和所有输入数据。
- custom resource provider：拥有自定义资源并确定如何处理和响应来自ROS的请求。custom resource provider必须提供template developer使用的服务令牌。
- ROS：在资源栈操作期间，向模板中指定的服务令牌发送请求，然后等待响应，再继续资源栈操作。

template developer和custom resource provider可以是同一人员或实体，但过程相同。自定义资源的步骤如下：

1. template developer在模板中定义自定义资源，该模板包含服务令牌和任何输入数据参数。根据自定义资源，输入数据可选，但服务令牌必选。
   服务令牌指定ROS发送请求发送的位置，例如：发送到MNS主题ARN或FC函数ARN。更多信息，请参见ALIYUN::ROS::CustomResource。服务令牌和输入数据的结构由custom resource provider定义。
2. 当您使用模板创建、更新或删除自定义资源时，ROS将向指定服务令牌发送请求。服务令牌无地域限制。
   在请求中，ROS包含请求类型和自定义资源向其发送请求的预签名URL等信息。更多信息，请参见自定义资源请求对象。
   以下示例列出ROS在请求中包含的内容：

```
{
   "RequestType" : "Create",
   "RequestId" : "unique id for this create request",
   "ResponseURL" : "pre-signed-url-for-create-response",
   "IntranetResponseURL" : "pre-signed-intranet-url-for-create-response",
   "ResourceType" : "Custom::MyCustomResourceType",
   "LogicalResourceId" : "name of resource in template",
   "StackId" : "stack id",
   "StackName" : "stack name",
   "ResourceOwnerId": "resource owner id",
   "CallerId": "caller id",
   "RegionId": "region id",
   "ResourceProperties" : {
      "key1" : "string",
      "key2" : [ "list" ],
      "key3" : { "key4" : "map" }
   }
}
```

3. custom resource provider处理ROS请求并向预签名URL返回SUCCESS或FAILED响应。custom resource provider提供采用JSON格式数据响应URL。
   在响应中，custom resource provider还可以包含template developer。例如：如果请求成功，响应可以包含输出数据，如果请求失败，相应可以包含错误消息。更多信息，请参见自定义资源响应对象。
   custom resource provider负责侦听和响应请求。例如：对于MNS主题通知，custom resource provider必须侦听并响应发送到特定主题ARN的通知。ROS在预签名URL位置等待并侦听响应。
   以下示例列出自定义资源在响应中可以包含的内容：

```
{
   "Status" : "SUCCESS",
   "RequestId" : "unique id for this create request (copied from request)",
   "LogicalResourceId" : "name of resource in template (copied from request)",
   "StackId" : "stack id (copied from request)",
   "PhysicalResourceId" : "required vendor-defined physical id that is unique for that
vendor",
   "Data" : {
      "keyThatCanBeUsedInGetAtt1" : "data for key 1",
      "keyThatCanBeUsedInGetAtt2" : "data for key 2"
   }
}
```

4. 获得SUCCESS响应后，ROS继续资源栈操作。如果收到FAILED响应或未返回任何响应，则操作失败。来自自定义资源的所有输出数据都由预签名URL响应返回。template developer可使用Fn::GetAtt函数检索该数据。

最佳实践

- 资源编排ROS之自定义资源（基础篇）
  通过函数计算（FC）实现复杂逻辑处理。
- 资源编排ROS之自定义资源（多云部署AWS篇）
  通过函数计算（FC）与AWS CloudFormation部署AWS资源，从而达到多云部署的目标。
- 资源编排ROS之自定义资源（多云部署Terraform篇）
  通过HTTP和HTTPS与Terraform部署AWS资源，从而达到多云部署的目标。

6.3.2. 资源参考

本文为您介绍自定义资源相关信息。

本章节提供以下内容的详细信息：

- 提供自定义资源时，发送到或发送自阿里云ROS的消息中使用的JSON请求和响应字段。
- custom resource provider响应资源栈创建、资源栈更新和资源栈删除时，对发出请求或进行响应所需的字段。

本章节包含内容如下：

6.3.2.1. 自定义资源参考

为您提供自定义资源时，发送到或发送自阿里云ROS的消息中使用的JSON请求和响应字段。custom resource provider响应资源栈创建、资源栈更新和资源栈删除时，对发出请求或进行响应所需的字段。

6.3.2.2. 自定义资源请求对象

通过本文档您可以了解开发人员模板请求类型和请求字段。

模板开发人员请求类型

Template developer使用阿里云ROS资源ALIYUN::ROS::CustomResource在模板中指定自定义资源。

在ALIYUN::ROS::CustomResource中，包含ServiceToken、Parameters、Timeout 3个属性。

属性名称	类型	是否必需	描述	约束
ServiceToken	String	是	服务令牌。由custom service provider向template developer提供。	目前支持FC（函数计算）Function、MNS（消息服务）Topic、MNS（消息服务）Queue、HTTP&HTTPS。服务令牌无地域限制。
Parameters	Map	否	要传递给custom service provider的参数。按照custom service provider提供的规范填写。	无
Timeout	Number	否	等待custom service provider响应的超时时间。	取值范围：1-43200 单位：秒 默认值：60

Custom Resource Provider请求字段

如下字段将以JSON请求形式从阿里云ROS发送到custom resource provider。

字段名称	类型	是否必需	描述	约束
RequestType	String	是	由阿里云ROS资源栈操作（创建、更新、删除资源栈）设置的请求类型。资源栈操作由包含自定义资源的资源栈的template developer启动。	取值： • Create • Update • Delete 更多信息，请参见自定义资源请求类型。
ResponseURL	String	是	预签名的公网URL。该URL接收custom resource provider到阿里云ROS的响应。	无
IntranetResponseURL	String	是	预签名的内网URL，阿里云ECS中可使用此URL。该URL接收custom resource provider到阿里云ROS的响应。	无
StackId	String	是	包含自定义资源的资源栈ID	无
StackName	String	是	包含自定义资源的资源栈名称	无
ResourceOwnerId	String	是	包含自定义资源的资源栈归属的阿里云账号ID（主账号）	无
CallerId	String	是	执行此次资源栈操作的账号ID（主账号或子账号）	无
RegionId	String	是	包含自定义资源的资源栈归属的地域	无
RequestId	String	是	请求的唯一ID。将StackId与RequestId结合可组成一个值，该值可用于唯一标识对特定自定义资源的请求。	无
ResourceType	String	是	阿里云ROS模板中模板开发人员选择的自定义资源的资源类型	最大支持68个字符，可包含字母、数字、下划线（_）、at（@）和短划线（-）。
LogicalResourceId	String	是	阿里云ROS模板中template developer选择的自定义资源名称（逻辑 ID）。 用于促进custom resource provider和template developer之间的通信。	无
PhysicalResourceId	String	是	custom resource provider定义的物理ID，该ID对于该提供程序是唯一的。	仅用于Update和Delete请求

字段名称	类型	是否必需	描述	约束
ResourceProperties	JSON object	是	资源属性。该字段包含template developer发送的Properties中Parameters对象的内容。其内容由custom resource provider定义。	无
OldResourceProperties	JSON object	否	在更新请求之前声明的资源属性	仅用于Update请求

6.3.2.3. 自定义资源响应对象

通过本文您可以了解自定义资源的响应字段。

Custom Resource Provider响应头

响应请求头必须包含如下字段：

- Content-type：取值 `"application/json"` 。
- Date：描述请求时间。GMT 格式，例如： `"Tue, 26 Nov 2019 08:46:44 GMT"` 。

Custom Resource Provider响应字段

以下是自定义资源提供程序在将JSON数据回调发送到URL（ResponseURL或InnerResponseURL） 时包含的属性。

响应正文的总大小不能超过 4096 字节。

- Status
  - custom resource provider为响应阿里云ROS生成的请求而发送的状态值。
  - 必须是SUCCESS或FAILED。
  - 是否必需：是。
  - 类型：String。
- Reason
  - 描述响应失败的原因。
  - 仅当 Status为FAILED 时有效。
  - 是否必需：如果Status为FAILED，则是必需的。否则，它是可选的。
  - 类型：String。
- PhysicalResourceId
  - 该值应是对于自定义资源供应商具有唯一性的标识符，最大大小为255字节。该值必须是非空字符串，并且在所有响应中对于相同资源必须相同。
  - 仅当Status为SUCCESS时有效。创建时，此值需传递。更新和删除时，此响应值应从请求中复制。
  - 是否必需：如果Status为SUCCESS，则是必需的。否则，它是可选的。
  - 类型：String。
- StackId
  - 包含自定义资源的资源栈的ID 。此响应值应从请求中复制。
  - 是否必需：是。

- 类型：String。

- RequestId
  - 请求的唯一 ID。 此响应值应从请求中复制。
  - 是否必需：是。
  - 类型：String。

- LogicalResourceId
  - 阿里云 ROS 模板中 template developer 选择的自定义资源名称（逻辑 ID）。此响应值应从请求中复制。
  - 是否必需：是。
  - 类型：String。

- Data
  - 可选。要在响应中发送的custom resource provider-defined的名称-值对。您可以使用 Fn::GetAtt 在模板中按名称访问此处提供的值。
  - 是否必需：否。
  - 类型：JSON object。

6.3.3. 请求类型

在 template developer 创建、更新或删除包含自定义资源的资源栈时，将在 阿里云 ROS 发送的供应商请求对象的 RequestType 字段中发送请求类型。

每种请求类型均包含一组与请求一起发送的特定字段，包括自定义资源提供者提供的响应的URL(ResponseURL或InnerResponseURL)。提供方必须在 Timeout(1-43200秒) 时间内提供 SUCCESS 或 FAILED 结果来响应 URL。超过Timeout时间后，请求将超时。每个结果还包含 阿里云 ROS 所需的一组特定字段。

此部分提供每种类型请求的请求字段和响应字段的相关信息，并提供相应示例。

在本章节中

- Create
- Update
- Delete

6.3.3.1. 自定义资源请求类型

当模板开发人员创建、更新或删除包含自定义资源的资源栈时，ROS会向供应商提供的服务发送请求，请求对象包含的RequestType字段表示请求类型。

每种请求类型均包含一组与请求一起发送的特定字段，包括自定义资源提供的响应URL（ResponseURL或InnerResponseURL）。供应商必须在Timeout（1-43200秒）时间内提供SUCCESS或FAILED结果来响应URL。超过Timeout时间后，请求将超时。每个结果还包含ROS所需的一组特定字段。

ROS为您提供如下请求类型：

- Create
- Update
- Delete

6.3.3.2. Create

当模板开发人员创建包含自定义资源的资源栈时，ROS将发送RequestType已设置为Create的自定义资源请求。

请求

创建请求包含以下字段：

- RequestType
  将为Create。
- ResponseURL
  预签名的公网URL。该URL接收custom resource provider到ROS的响应。
- IntranetResponseURL
  预签名的内网URL，ECS中可使用此URL。该URL接收custom resource provider到ROS的响应。
- StackId
  包含自定义资源的资源栈的ID 。
- StackName
  包含自定义资源的资源栈的名称。
- ResourceOwnerId
  包含自定义资源的资源栈归属的阿里云账号ID（阿里云主账号）。
- CallerId
  执行此次资源栈操作的账号ID（阿里云主账号或RAM用户）。
- RegionId
  包含自定义资源的资源栈归属的地域。
- RequestId
  请求的唯一ID。
- ResourceType
  ROS模板中模板开发人员选择的自定义资源的资源类型。自定义资源类型名称最长为68个字符，可包含英文字符、数字、下划线（_）、at（@）和短划线（-）。
- LogicalResourceId
  ROS模板中模板开发人员选择的自定义资源名称（逻辑 ID）。
- ResourceProperties
  该字段包含template developer发送的Properties中Parameters对象的内容。其内容由custom resource provider定义。

示例：

```
{
   "RequestType" : "Create",
   "RequestId" : "unique id for this create request",
   "ResponseURL" : "pre-signed-url-for-create-response",
   "IntranetResponseURL" : "pre-signed-intranet-url-for-create-response",
   "ResourceType" : "Custom::MyCustomResourceType",
   "LogicalResourceId" : "name of resource in template",
   "StackId" : "stack id",
   "StackName" : "stack name",
   "ResourceOwnerId": "resource owner id",
   "CallerId": "caller id",
   "RegionId": "region id",
   "ResourceProperties" : {
      "key1" : "string",
      "key2" : [ "list" ],
      "key3" : { "key4" : "map" }
   }
}
```

响应

成功

创建请求成功时，供应商提供的服务必须向ROS发送包含以下字段的响应：

- Status
  必须为SUCCESS。
- RequestId
  请求的唯一ID。此响应值应从请求中复制。
- LogicalResourceId
  ROS模板中模板开发人员选择的自定义资源名称（逻辑 ID）。此响应值应从请求中复制。
- StackId
  包含自定义资源的资源栈的ID 。此响应值应从请求中复制。
- PhysicalResourceId
  该值应是对于自定义资源供应商具有唯一性的标识符，最大为255字节。该值必须是非空字符串，并且在所有响应中对于相同资源必须相同。
- Data
  可选。要在响应中发送的custom resource provider-defined的名称-值对。您可以使用Fn::GetAtt在模板中按名称访问此处提供的值。

示例：

```
{
   "Status" : "SUCCESS",
   "RequestId" : "unique id for this create request (copied from request)",
   "LogicalResourceId" : "name of resource in template (copied from request)",
   "StackId" : "stack id (copied from request)",
   "PhysicalResourceId" : "required vendor-defined physical id that is unique for that vend
or",
   "Data" : {
      "keyThatCanBeUsedInGetAtt1" : "data for key 1",
      "keyThatCanBeUsedInGetAtt2" : "data for key 2"
   }
}
```

失败

创建请求失败时，供应商提供的服务必须向ROS发送包含以下字段的响应：

- Status
  必须为FAILED。
- Reason
  描述响应失败的原因。
- RequestId
  请求的唯一ID。 此响应值应从请求中复制。
- LogicalResourceId
  ROS模板中模板开发人员选择的自定义资源名称（逻辑 ID）。此响应值应从请求中复制。
- StackId
  包含自定义资源的资源栈的ID。此响应值应从请求中复制。

示例：

```
{
   "Status" : "FAILED",
   "Reason" : "Required failure reason string",
   "RequestId" : "unique id for this create request (copied from request)",
   "LogicalResourceId" : "name of resource in template (copied from request)",
   "StackId" : "stack id (copied from request)"
}
```

6.3.3.3. Update

在对模板内的自定义资源属性进行更改时，ROS将发送RequestType已设置为Update的自定义资源请求。因此，自定义资源代码不必检测更改，因为它知道其属性在调用Update后已更改。

请求

更新请求包含以下字段：

- RequestType
  将为Update。
- ResponseURL
  预签名的公网URL。该URL接收custom resource provider到ROS的响应。
- IntranetResponseURL

预签名的内网URL，ECS中可使用此URL。该URL接收custom resource provider到ROS的响应。

- StackId
  包含自定义资源的资源栈的ID 。
- StackName
  包含自定义资源的资源栈的名称。
- ResourceOwnerId
  包含自定义资源的资源栈归属的阿里云账号ID（阿里云主账号）。
- CallerId
  执行此次资源栈操作的账号ID（阿里云主账号或RAM用户）。
- RegionId
  包含自定义资源的资源栈归属的地域。
- RequestId
  请求的唯一ID。
- ResourceType
  ROS模板中模板开发人员选择的自定义资源的资源类型。自定义资源类型名称最长为68个字符，可包含英文字符、数字、下划线（_）、at（@）和短划线（-）。
- LogicalResourceId
  ROS模板中template developer选择的自定义资源名称（逻辑 ID）。
- PhysicalResourceId
  custom resource provider定义的物理 ID，该ID对于该提供程序是唯一的。
- ResourceProperties
  该字段包含模板开发人员发送的Properties中Parameters对象的内容。其内容由custom resource provider定义。
- OldResourceProperties
  该字段包含模板开发人员以前定义的Properties中Parameters对象的内容。

示例

```
{
   "RequestType" : "Update",
   "RequestId" : "unique id for this update request",
   "ResponseURL" : "pre-signed-url-for-update-response",
   "IntranetResponseURL" : "pre-signed-intranet-url-for-create-response",
   "ResourceType" : "Custom::MyCustomResourceType",
   "LogicalResourceId" : "name of resource in template",
   "PhysicalResourceId" : "custom resource provider-defined physical id",
   "StackId" : "stack id",
   "StackName" : "stack name",
   "ResourceOwnerId": "resource owner id",
   "CallerId": "caller id",
   "RegionId": "region id",
   "ResourceProperties" : {
      "key1" : "new-string",
      "key2" : [ "new-list" ],
      "key3" : { "key4" : "new-map" }
   },
   "OldResourceProperties" : {
      "key1" : "string",
      "key2" : [ "list" ],
      "key3" : { "key4" : "map" }
   }
}
```

响应

成功

更新请求成功时，供应商提供的服务必须向ROS发送包含以下字段的响应：

- Status
  必须为SUCCESS。
- RequestId
  请求的唯一ID。 此响应值应从请求中复制。
- LogicalResourceId
  ROS模板中模板开发人员选择的自定义资源名称（逻辑 ID）。此响应值应从请求中复制。
- StackId
  包含自定义资源的资源栈的ID。此响应值应从请求中复制。
- PhysicalResourceId
  该值不可变。此响应值应从请求中复制。
- Data
  可选。要在响应中发送的custom resource provider-defined的名称-值对。您可以使用Fn::GetAtt在模板中按名称访问此处提供的值。

示例

```
{
   "Status" : "SUCCESS",
   "RequestId" : "unique id for this update request (copied from request)",
   "LogicalResourceId" : "name of resource in template (copied from request)",
   "StackId" : "stack id (copied from request)",
   "PhysicalResourceId" : "custom resource provider-defined physical id",
   "Data" : {
      "keyThatCanBeUsedInGetAtt1" : "data for key 1",
      "keyThatCanBeUsedInGetAtt2" : "data for key 2"
   }
}
```

失败

更新请求失败时，必须向 ROS 发送包含以下字段的响应：

- Status
  必须为FAILED。
- Reason
  描述响应失败的原因。
- RequestId
  请求的唯一ID。此响应值应从请求中复制。
- LogicalResourceId
  ROS模板中模板开发人员选择的自定义资源名称（逻辑 ID）。此响应值应从请求中复制。
- StackId
  包含自定义资源的资源栈的ID。此响应值应从请求中复制。

示例

```
{
   "Status" : "FAILED",
   "Reason" : "Required failure reason string",
   "RequestId" : "unique id for this update request (copied from request)",
   "LogicalResourceId" : "name of resource in template (copied from request)",
   "StackId" : "stack id (copied from request)"
}
```

6.3.3.4. Delete

当模板开发人员删除包含自定义资源的堆栈时，ROS将发送RequestType设置为Delete的自定义资源请求。要成功删除带自定义资源的资源栈，custom resource provider必须成功响应删除请求。

请求

删除请求包含以下字段：

- RequestType
  将为Delete。
- ResponseURL
  预签名的公网URL。该URL接收custom resource provider到ROS的响应。
- IntranetResponseURL
  预签名的内网URL，ECS中可使用此URL。该URL接收custom resource provider到ROS的响应。

- StackId
  包含自定义资源的资源栈的ID。
- StackName
  包含自定义资源的资源栈的名称。
- ResourceOwnerId
  包含自定义资源的资源栈归属的阿里云账号ID（阿里云主账号）。
- CallerId
  执行此次资源栈操作的账号ID（阿里云主账号或RAM用户）。
- RegionId
  包含自定义资源的资源栈归属的地域。
- RequestId
  请求的唯一ID。
- ResourceType
  ROS模板中模板开发人员选择的自定义资源的资源类型。自定义资源类型名称最长为68个字符，可包含英文字符、数字、下划线（_）、at（@）和短划线（-）。
- LogicalResourceId
  ROS模板中template developer选择的自定义资源名称（逻辑ID）。
- PhysicalResourceId
  custom resource provider定义的物理ID，该ID对于该提供程序是唯一的。
- ResourceProperties
  该字段包含模板开发人员发送的Properties中Parameters对象的内容。其内容由custom resource provider定义。

示例：

```
{
   "RequestType" : "Delete",
   "RequestId" : "unique id for this delete request",
   "ResponseURL" : "pre-signed-url-for-delete-response",
   "IntranetResponseURL" : "pre-signed-intranet-url-for-create-response",
   "ResourceType" : "Custom::MyCustomResourceType",
   "LogicalResourceId" : "name of resource in template",
   "PhysicalResourceId" : "custom resource provider-defined physical id",
   "StackId" : "stack id",
   "StackName" : "stack name",
   "ResourceOwnerId": "resource owner id",
   "CallerId": "caller id",
   "RegionId": "region id",
   "ResourceProperties" : {
      "key1" : "string",
      "key2" : [ "list" ],
      "key3" : { "key4" : "map" }
   }
}
```

响应

成功

删除请求成功时，供应商提供的服务必须向ROS发送包含以下字段的响应：

- Status

必须为SUCCESS。

- RequestId
  请求的唯一ID。 此响应值应从请求中复制。
- LogicalResourceId
  ROS模板中模板开发人员选择的自定义资源名称（逻辑ID）。此响应值应从请求中复制。
- StackId
  包含自定义资源的资源栈的ID。此响应值应从请求中复制。
- PhysicalResourceId
  该值应是对于自定义资源供应商具有唯一性的标识符，最大为255字节。该值必须是非空字符串，并且在所有响应中对于相同资源必须相同。

示例

```
{
   "Status" : "SUCCESS",
   "RequestId" : "unique id for this delete request (copied from request)",
   "LogicalResourceId" : "name of resource in template (copied from request)",
   "StackId" : "stack id (copied from request)",
   "PhysicalResourceId" : "custom resource provider-defined physical id"
}
```

失败

删除请求失败时，供应商提供的服务必须向ROS发送包含以下字段的响应：

- Status
  必须为FAILED。
- Reason
  描述响应失败的原因。
- RequestId
  请求的唯一ID。 此响应值应从请求中复制。
- LogicalResourceId
  ROS模板中模板开发人员选择的自定义资源名称（逻辑ID）。此响应值应从请求中复制。
- StackId
  包含自定义资源的资源栈的ID。此响应值应从请求中复制。

示例

```
{
   "Status" : "FAILED",
   "Reason" : "Required failure reason string",
   "RequestId" : "unique id for this delete request (copied from request)",
   "LogicalResourceId" : "name of resource in template (copied from request)",
   "StackId" : "stack id (copied from request)"
}
```

6.4. 数据源资源

数据源资源（DataSource）用于查询云服务的资源数据。数据源资源可以被其他资源引用，也可以在输出（Outputs）中被引用。数据源资源和普通资源除了作用不同，支持的功能（例如：引用、依赖、更新等）完全相同。

应用场景

- 将动态查询结果作为创建资源的输入
  ROS根据指定的条件动态查询结果，然后将此结果作为创建其他资源的输入属性。
- 在模板输出中呈现资源详情
  在模板资源中创建数据源资源后，在模板输出中即可引用该数据源资源，以呈现资源详情。

将动态查询结果作为创建资源的输入

ROS根据指定的条件动态查询结果，然后将此结果作为创建其他资源的输入属性。

以下示例模板中指定了多个参数，用于查询参数取值列表，ROS将动态查询结果作为创建ECS实例的输入属性。

- CPU核数和内存动态：用于查询可用的ECS实例规格列表，ROS默认选择列表中第一个规格创建ECS实例。
- 镜像名称：用于动态查询镜像列表，ROS默认选择列表中第一个镜像ID创建ECS实例。

```
{
  "ROSTemplateFormatVersion": "2015-09-01",
  "Parameters": {
    "ZoneId": {
      "Type": "String",
      "AssociationProperty": "ALIYUN::ECS::ZoneId"
    },
    "VpcId": {
      "Type": "String",
      "AssociationProperty": "ALIYUN::ECS::VPC::VPCId"
    },
    "VSwitchId": {
      "Type": "String",
      "AssociationProperty": "ALIYUN::ECS::VSwitch::VSwitchId",
      "AssociationPropertyMetadata": {
        "VpcId": "${VpcId}",
        "ZoneId": "${ZoneId}"
      }
    },
    "SecurityGroupId": {
      "Type": "String",
      "AssociationProperty": "ALIYUN::ECS::SecurityGroup::SecurityGroupId",
      "AssociationPropertyMetadata": {
        "VpcId": "${VpcId}"
      }
    }
  },
  "Resources": {
    "DS-RecommendInstanceTypes": {
      "Type": "DATASOURCE::ECS::RecommendInstanceTypes",
      "Properties": {
        "Cores": 1,
        "Memory": 1
      }
    },
    "DS-Images": {
      "Type": "DATASOURCE::ECS::Images",
      "Properties": {
        "ImageName": "CentOS8*",
```

```
        "InstanceType": {
          "Fn::Select": [0, { "Ref": "DS-RecommendInstanceTypes" }]
        }
      }
    },
    "Instance": {
      "Type": "ALIYUN::ECS::Instance",
      "Properties": {
        "InstanceName": "MyInstance",
        "ImageId": { "Fn::Select": [0, { "Ref": "DS-Images" }] },
        "InstanceType": {
          "Fn::Select": [0, { "Ref": "DS-RecommendInstanceTypes" }]
        },
        "ZoneId": {"Ref": "ZoneId"},
        "VSwitchId": {"Ref": "VSwitchId"},
        "SecurityGroupId": {"Ref": "SecurityGroupId"},
        "SystemDiskCategory": "cloud_efficiency"
      }
    }
  },
  "Outputs": {
    "InstanceId": {
        "Value": {"Ref": "Instance" }
    }
  }
}
```

模板说明：

- 参数（Parameters）中定义了4个参数：`ZoneId`、`VpcId`、`VSwitchId`和`SecurityGroupId`。每个参数都配置了`AssociationProperty`，以便在ROS控制台的参数选择界面查询参数的取值列表。
- 资源（Resources）中定义了3个资源，包含2个数据源资源和1个普通资源。
  - 逻辑ID为`DS-RecommendInstanceTypes`的数据源资源根据指定的1核1 G的条件，查询符合条件的ECS实例规格。
  - 逻辑ID为`DS-Images`的数据源资源使用`{ "Fn::Select": [0, { "Ref": "RecommendInstanceTypes" } }`获取ECS实例规格列表中的第1个规格作为`InstanceType`输入，并指定镜像名称以CentOS开头，查询符合条件的镜像。
  - 逻辑ID为`Instance`的普通资源将ROS获取到的实例规格和镜像作为输入，创建ECS实例。
- 输出（Outputs）中定义了1个输出变量`InstanceId`。

在模板输出中呈现资源详情

在模板资源中创建数据源资源后，在模板输出中即可引用该数据源资源，以呈现资源详情。

以下示例模板中，创建VPC后，ROS将通过源资源查询VPC的详细数据，并在输出中呈现VPC详情。

```
{
  "ROSTemplateFormatVersion": "2015-09-01",
  "Resources": {
    "Vpc": {
      "Type": "ALIYUN::ECS::VPC",
      "Properties": {
        "VpcName": "MyVpc",
        "CidrBlock": "172.16.0.0/12"
      }
    },
    "DS-Vpcs": {
      "Type": "DATASOURCE::VPC::Vpcs",
      "Properties": {
        "VpcIds": [{ "Ref": "Vpc" }]
      }
    }
  },
  "Outputs": {
    "VpcData": {
      "Value": {
        "Fn::Select": [
          0,
          {
            "Fn::GetAtt": ["DS-Vpcs", "Vpcs"]
          }
        ]
      }
    }
  }
}
```

模板说明如下：

- 资源（Resources）中定义了2个资源，包含1个普通资源和1个数据源资源。
  - 逻辑ID为 `Vpc` 的普通资源指定VPC名称和网段，用于创建1个VPC。
  - 逻辑ID为 `DS-Vpcs` 的数据源资源将已创建的VPC ID作为输入，查询VPC详情。
- 输出（Outputs）中定义了1个输出变量 `VpcData` ，取值为数据源资源 `DS-Vpcs` 的 `Vpcs` 属性值的第1个值。

相关文档

- GetResourceType
- ListResourceTypes

6.5. 预估资源价格

当模板中使用的资源需要收费时，您需要了解资源价格。资源编排ROS（Resource Orchestration Service）支持通过控制台和API对您的资源进行询价，从而预估模板中资源的价格。

预估方式

您可以通过以下两种方式预估资源价格：

- 资源编排控制台：当您使用模板创建资源栈时，如果模板中有支持询价的资源，ROS将在**检查并确认**环节给出这些资源的价格预估（询价的价格仅供参考，具体价格以实际支付为准）。
- API：您可以调用GetTemplateEstimateCost接口，预估模板中资源的价格。

支持询价的资源类型

> **说明** 资源付费方式与您在模板中设置的付费类型参数有关。例如：如果您将ALIYUN::ECS::Instance的付费类型（InstanceChargeType）设置为PrePaid，则创建的ECS实例付费方式为预付费。

云服务	资源类型
应用型负载均衡ALB	ALIYUN::ALB::LoadBalancer
云数据库专属集群MyBase	ALIYUN::CDDC::DedicatedHost
云企业网CEN	ALIYUN::CEN::CenBandwidthPackage
容器服务Kubernetes版ACK	• ALIYUN::CS::KubernetesCluster • ALIYUN::CS::ManagedEdgeKubernetesCluster • ALIYUN::CS::ManagedKubernetesCluster **说明** ROS也支持预估Kubernetes托管版集群的集群管理费。 • ALIYUN::CS::ServerlessKubernetesCluster **说明** ROS仅支持预估负载均衡SLB、云服务器ECS、NAT网关和弹性公网IP资源的价格。
云原生分布式数据库PolarDB-X	ALIYUN::DRDS::DrdsInstance
弹性容器实例ECI	• ALIYUN::ECI::ContainerGroup • ALIYUN::ECI::ImageCache
云服务器ECS	• ALIYUN::ECS::DedicatedHost • ALIYUN::ECS::Disk • ALIYUN::ECS::Instance • ALIYUN::ECS::InstanceGroup
弹性高性能计算EHPC	ALIYUN::EHPC::Cluster
阿里云Elasticsearch	ALIYUN::ElasticSearch::Instance
阿里云E-MapReduce	ALIYUN::EMR::Cluster
边缘节点服务ENS	ALIYUN::ENS::Instance

云服务	资源类型
云市场Marketplace	ALIYUN::MarketPlace::Order
云数据库Memcache版	ALIYUN::Memcache::Instance
云数据库MongoDB版	• ALIYUN::MONGODB::Instance • ALIYUN::MONGODB::ServerlessInstance • ALIYUN::MONGODB::ShardingInstance
云数据库PolarDB	ALIYUN::POLARDB::DBCluster
云数据库Redis版	ALIYUN::REDIS::Instance
阿里云关系型数据库RDS	• ALIYUN::RDS::DBInstance • ALIYUN::RDS::PrepayDBInstance • ALIYUN::RDS::ReadOnlyDBInstance
资源编排服务ROS	ALIYUN::ROS::Stack
Serverless应用引擎SAE	ALIYUN::SAE::Application
智能接入网关SAG	ALIYUN::SAG::SmartAccessGateway
负载均衡SLB	ALIYUN::SLB::LoadBalancer
专有网络VPC	• ALIYUN::VPC::AnycastEIP • ALIYUN::VPC::CommonBandwidthPackage • ALIYUN::VPC::CustomerGateway • ALIYUN::VPC::EIP • ALIYUN::VPC::EIPPro • ALIYUN::VPC::NatGateway • ALIYUN::VPC::Ipv6Gateway • ALIYUN::VPC::VpnGateway
云原生数据仓库AnalyticDB PostgreSQL版	• ALIYUN::GPDB::DBInstance • ALIYUN::GPDB::ElasticDBInstance
API网关API Gateway	ALIYUN::ApiGateway::Instance
数据传输服务DTS	• ALIYUN::DTS::SubscriptionInstance • ALIYUN::DTS::SynchronizationJob • ALIYUN::DTS::MigrationJob
消息队列Kafka版	ALIYUN::KAFKA::Instance
文件存储NAS	ALIYUN::NAS::FileSystem

云服务	资源类型
云数据库ClickHouse	ALIYUN::ClickHouse::DBCluster
全球加速GA	ALIYUN::GA::Accelerator
时序数据库TSDB	ALIYUN::TSDB::HiTSDBInstance
微服务引擎MSE	ALIYUN::MSE::Cluster
Web应用防火墙WAF	ALIYUN::WAF::Instance
云原生数据仓库AnalyticDB MySQL版	ALIYUN::ADB::DBCluster

6.6. 模板快速入门

模板用于创建资源栈，是描述基础设施和架构的蓝图。您需要在模板中声明各类云服务的资源，然后通过ROS部署资源。

背景信息

当您了解ROS的模板结构后，可以尝试编写您的第一个模板。关于模板结构的更多信息，请参见模板结构说明。

本文从简入难，为您介绍如何编写和测试一个简单的模板（创建VPC），然后深入讲解如何在模板中定义多个资源及其依赖关系、在模板中定义参数，从而满足您在多个部署场景的需求，具体如下：

- 编写和测试一个简单的模板
  i. 编写模板
  ii. 测试模板
- 在模板中定义多个资源及其依赖关系
- 在模板中定义参数取值和参数属性

编写模板

编写模板最为重要的部分是在资源（Resources）中声明需要创建的资源，您可以在资源类型索引中查看ROS支持的所有资源类型，然后单击特定资源类型查看该资源类型支持的属性和返回值信息。

1. 根据部署场景查询资源类型详情。
   i. 访问资源类型索引，根据部署场景查询资源类型（例如：创建VPC需要用到资源类型ALIYUN::ECS::VPC）。
   ii. 访问指定资源类型（例如：ALIYUN::ECS::VPC），查询属性详情。

   资源类型文档中为每个属性定义了“类型”、“必须”、“允许更新”、“描述”、“约束”等信息，其中：
   - 若“必须”为是，则要求必须在模板Resources的Properties中声明该属性；反之，则选填。
   - 若“允许更新”为是，则意味着新模板中可以修改此属性，再使用新模板去更新资源栈以达到更新云资源属性的目的；反之，则不允许更新。
2. 在资源（Resources）中声明需要创建的资源。

   您可以定义专有网络名称（VpcName）和专有网络网段（CidrBlock），编写Resources模板示例代码段。

```
"Resources": {
    "VPC": {
      "Type": "ALIYUN::ECS::VPC",
      "Properties": {
        "VpcName": "myvpc",
        "CidrBlock": "192.168.0.0/16"
      }
    }
  }
```

3. 在输出（Outputs）中定义使用模板创建资源栈后的输出内容。

   您可以定义专有网络ID（VpcId），编写Outputs模板示例代码段。

```
"Outputs": {
    "VpcId": {
      "Value": { "Ref": "VPC" }
    },
    "VRouterId": {
      "Value": { "Fn::GetAtt": ["VPC", "VRouterId"] }
    }
  }
```

4. 根据模板结构，编写完整的模板。

   更多信息，请参见模板结构说明。
   创建VPC的完整模板示例代码如下：

```
{
  "ROSTemplateFormatVersion": "2015-09-01",
  "Resources": {
    "VPC": {
      "Type": "ALIYUN::ECS::VPC",
      "Properties": {
        "VpcName": "myvpc",
        "CidrBlock": "192.168.0.0/16"
      }
    }
  },
  "Outputs": {
    "VpcId": {
      "Value": { "Ref": "VPC" }
    },
    "VRouterId": {
      "Value": { "Fn::GetAtt": ["VPC", "VRouterId"] }
    }
  }
}
```

测试模板

编写模板完成后，您可以使用模板创建资源栈，测试通过该模板是否可以创建预期的资源。

1. 登录资源编排控制台。
2. 创建资源栈。
   i. 在左侧导航栏，单击**资源栈**。

ii. 在页面左上角的地域下拉列表，选择资源栈的所在地域，例如：华北2（北京）。

iii. 在**资源栈列表**页面，单击**创建资源栈**，然后在下拉列表中选择**使用新资源（标准）**。

iv. 在**选择模板**页面，在**指定模板**区域单击**选择已有模板**、选择**模板录入方式**为**输入模板**，然后在**模板内容**区域的**ROS**页签输入编写模板章节中编写的JSON格式的模板，最后单击**下一步**。

v. 在**配置模板参数**页面，输入资源栈名称，然后单击**下一步**。

vi. 在**配置资源栈**页面，配置**资源栈策略**、**失败时回滚**、**超时设置**、**删除保护**、**RAM角色**、**标签**和**资源组**，然后单击**下一步**。

vii. 在**检查并确认**页面，单击**预览模板资源**，然后在**预览**对话框查看经过ROS校验的模板中的资源名称、资源类型和资源属性，最后单击**确定**。

viii. 在**检查并确认**页面，单击**创建**。

3. 查看资源栈。

i. 在资源栈管理页面，单击**事件**页签，查看模板中不同资源的事件列表。

ii. 单击**资源**页签，查看已创建的资源详情。

> **说明** 您也可以单击模板资源ID，然后在资源相应的控制台查看更多资源信息，进一步确认资源是否符合预期。

iii. 单击**输出**页签，查看在模板Outputs中定义的输出。

在模板中定义多个资源及其依赖关系

了解如何编写基础的VPC资源相关模板后，您可能需要根据实际部署场景定义多个资源，以及资源之间的依赖关系。例如：交换机vSwitch依赖于VPC，您需要在特定的VPC中创建vSwitch。通过模板定义VPC、vSwitch资源及其依赖关系，然后创建资源栈，可以满足更为复杂的部署场景。

1. 使用Fn::GetAtt函数获取资源输出属性值。

   例如：假设Resources中定义了VPC，可以通过 `{"Fn::GetAtt": ["VPC", "VpcId"]}` 获取VPC资源的输出属性VpcId。

2. 使用Ref函数获取资源ID或参数值。

   例如：假设Resources中定义了VPC，可以通过 `{"Ref": "VPC"}` 引用VPC资源ID。

   > **说明**
   > - `{"Ref": "VPC"}` 与 `{"Fn::GetAtt": ["VPC", "VpcId"]}` 作用相同，但前者更为便捷。
   > - Ref和Fn::GetAtt函数隐式声明了资源间的依赖关系，您也可以通过DependsOn属性显式声明资源间的依赖关系。

3. 优化模板。

   以下模板示例，新增声明一个vSwitch，其可用区为cn-beijing-f、名称为myvsw、CidrBlock为192.168.0.0/24，并引用VPC。此外，在Outputs中定义了vSwitch的输出为交换机ID。

```
{
  "ROSTemplateFormatVersion": "2015-09-01",
  "Resources": {
    "VPC": {
      "Type": "ALIYUN::ECS::VPC",
      "Properties": {
        "VpcName": "myvpc",
        "CidrBlock": "192.168.0.0/16"
      }
    },
    "VSwitch": {
      "Type": "ALIYUN::ECS::VSwitch",
      "Properties": {
        "VpcId": { "Ref": "VPC" },
        "ZoneId": "cn-beijing-f",
        "VSwitchName": "myvsw",
        "CidrBlock": "192.168.0.0/24"
      }
    }
  },
  "Outputs": {
    "VpcId": {
      "Value":  "Fn::GetAtt": ["VPC", "VpcId"] }
    },
    "VRouterId": {
      "Value": { "Fn::GetAtt": ["VPC", "VRouterId"] }
    },
    "VSwitchId": {
      "Value": { "Ref": "VSwitch" }
    }
  }
}
```

在模板中定义参数取值和参数属性

参数（Parameters）可以提高模板的灵活性和可复用性，您可以通过定义参数取值和参数属性实现动态展示参数取值列表、为参数分组等需求，具体如下：

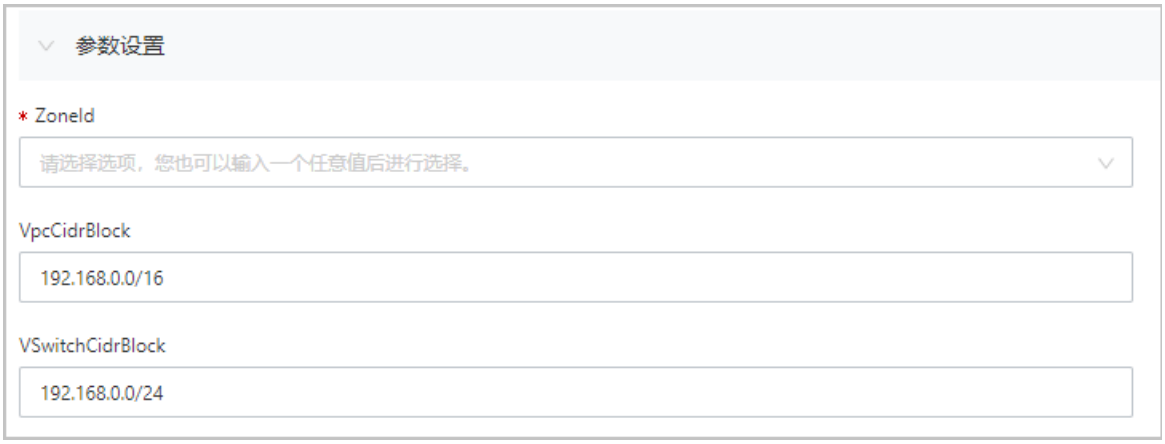
- **定义参数**
  在模板中为资源属性指定固定值的方式比较便捷，但是不够灵活。例如：ZoneId为cn-beijing-f，只能在北京地域创建资源栈，如果要更换地域需手动修改模板中的ZoneId取值。此时您可以将常用的或共同的属性提取出来定义为参数，以便在不修改模板的前提下，通过指定不同的参数来创建不同属性的资源。模板中定义的参数和模板示例代码如下：
  - 模板中定义的多个参数

参数	说明
`ZoneId`	被vSwitch的ZoneId属性引用。
`VpcCidrBlock`	默认值为192.168.0.0/16，被VPC的CidrBlock属性引用。
`VSwitchCidrBlock`	默认值为192.168.0.0/24，被vSwitch的CidrBlock属性引用

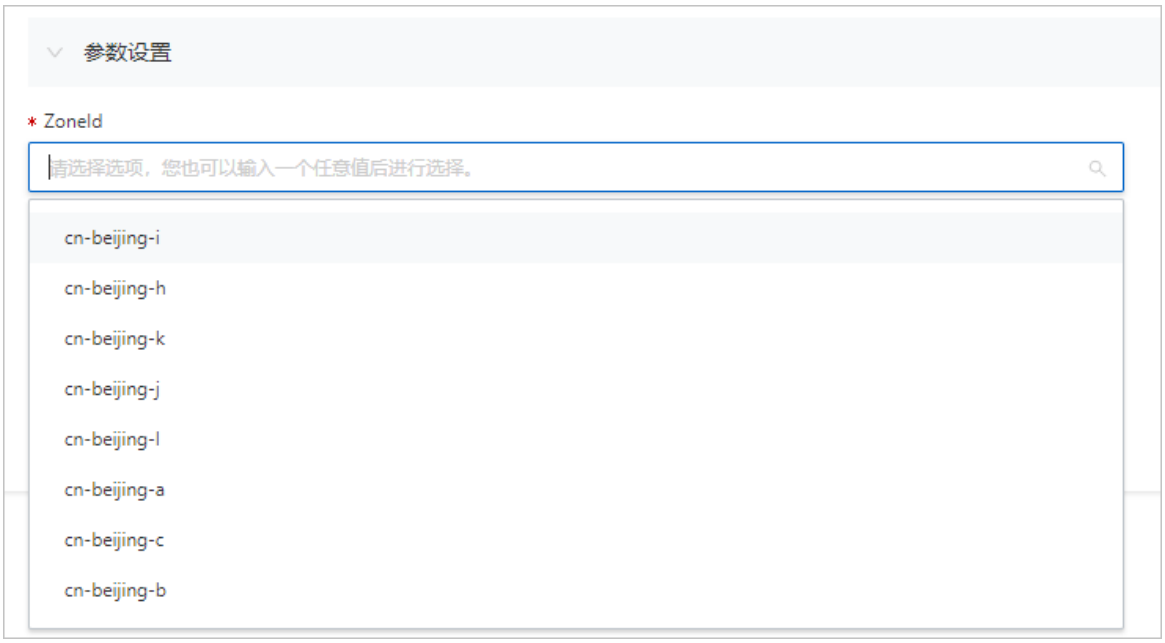
  - 模板示例代码

```
{
  "ROSTemplateFormatVersion": "2015-09-01",
  "Parameters": {
    "ZoneId": {
      "Type": "String"
    },
    "VpcCidrBlock": {
      "Type": "String",
      "Default": "192.168.0.0/16"
    },
    "VSwitchCidrBlock": {
      "Type": "String",
      "Default": "192.168.0.0/24"
    }
  },
  "Resources": {
    "VPC": {
      "Type": "ALIYUN::ECS::VPC",
      "Properties": {
        "VpcName": "myvpc",
        "CidrBlock": { "Ref": "VpcCidrBlock" }
      }
    },
    "VSwitch": {
      "Type": "ALIYUN::ECS::VSwitch",
      "Properties": {
        "VpcId": { "Ref": "VPC" },
        "ZoneId": { "Ref": "ZoneId" },
        "VSwitchName": "myvsw",
        "CidrBlock": { "Ref": "VSwitchCidrBlock" }
      }
    }
  },
  "Outputs": {
    "VpcId": {
      "Value": { "Ref": "VPC" }
    },
    "VRouterId": {
      "Value": { "Fn::GetAtt": ["VPC", "VRouterId"] }
    },
    "VSwitchId": {
      "Value": { "Ref": "VSwitch" }
    }
  }
}
```

使用模板示例代码创建资源栈时，您可以在资源编排控制台根据需要灵活设置参数取值。



其中，ROS将分析模板中参数和资源属性的关联关系，获取参数的取值范围。例如：模板中的ZoneId参数关联了vSwitch资源的ZoneId属性，ROS将获取支持vSwitch的可用区列表，并在控制台展示。



- **动态设置取值列表**

  参数（Parameters）支持多个属性，您可以通过定义属性，在控制台灵活呈现参数属性及取值。模板中定义的参数属性和模板示例代码如下：

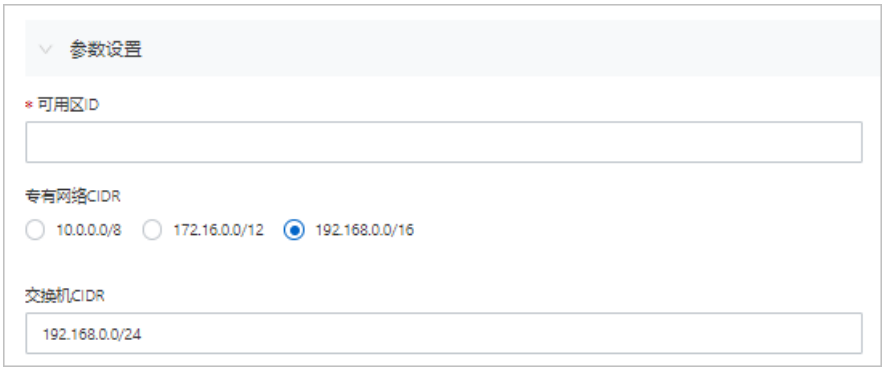
  - 参数中定义的多个属性

属性	说明
AllowedValues	定义取值列表，ROS控制台会为参数显示该列表。例如：VpcCidrBlock取值为10.0.0.0/8、172.16.0.0/12和192.168.0.0/16。
Label	定义参数别名，ROS控制台将显示此别名。例如：ZoneId显示为可用区ID。

- 模板示例代码

```
{
  "ROSTemplateFormatVersion": "2015-09-01",
  "Parameters": {
    "ZoneId": {
      "Type": "String",
      "Label": "可用区ID"
    },
    "VpcCidrBlock": {
      "Type": "String",
      "Label": "专有网络CIDR",
      "Default": "192.168.0.0/16",
      "AllowedValues": ["10.0.0.0/8", "172.16.0.0/12", "192.168.0.0/16"]
    },
    "VSwitchCidrBlock": {
      "Type": "String",
      "Label": "交换机CIDR",
      "Default": "192.168.0.0/24"
    }
  }
}
```

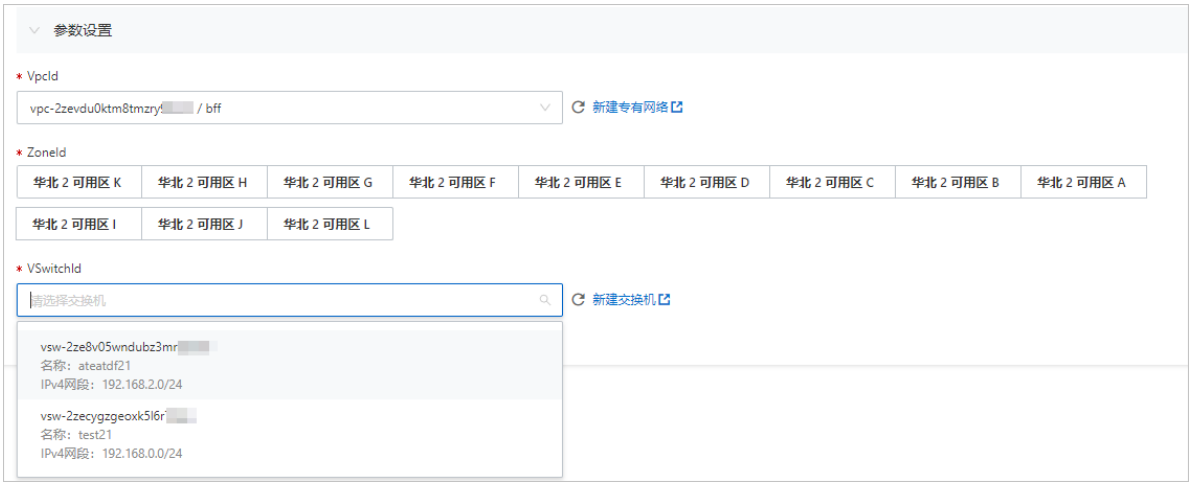
使用模板示例代码创建资源栈时，您可以在资源编排控制台直接选择参数取值。



- **基于参数关联关系动态呈现参数**
  定义参数的关联属性（AssociationProperty和AssociationPropertyMetadata）后，ROS控制台将动态查询参数的取值列表。更多信息，请参见AssociationProperty和AssociationPropertyMetadata。
  例如：假设模板中要创建ECS实例，其VpcId和vSwitchId作为参数传入，ROS控制台可以自动显示VpcId和vSwitchId的取值下拉列表。模板示例代码如下：

```
{
  "ROSTemplateFormatVersion": "2015-09-01",
  "Parameters": {
    "VpcId": {
      "Type": "String",
      "AssociationProperty": "ALIYUN::ECS::VPC::VPCId"
    },
    "ZoneId": {
      "Type": "String",
      "AssociationProperty": "ALIYUN::ECS::ZoneId"
    },
    "VSwitchId": {
      "Type": "String",
      "AssociationProperty": "ALIYUN::ECS::VSwitch::VSwitchId",
      "AssociationPropertyMetadata": {
        "ZoneId": "${ZoneId}",
        "VpcId": "${VpcId}"
      }
    }
  }
}
```

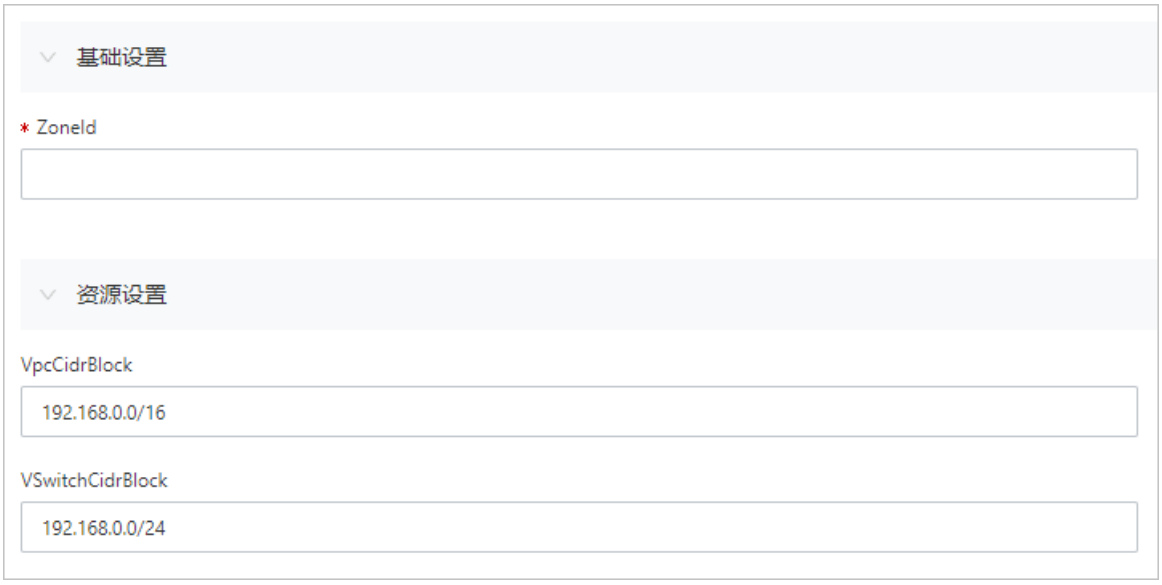
使用模板示例代码创建资源栈时，您可以在资源编排控制台将动态呈现参数。



- **集中设置同类参数**
  元数据（Metadata）支持为参数分组。当模板中的参数较多时，可以根据参数的特征为其分组，以便在控制台集中配置参数。
  例如：将模板中的ZoneId、VpcCidrBlock、`VSwitchCidrBlock`参数分组到基础设置和资源设置，其中基础设置为ZoneId，资源设置为VpcCidrBlock、`VSwitchCidrBlock`。您可以在Metadata的ParameterGroups中定义这两个分组。模板示例代码如下：

```
{
  "ROSTemplateFormatVersion": "2015-09-01",
  "Parameters": {
    "ZoneId": {
      "Type": "String"
    },
    "VpcCidrBlock": {
      "Type": "String",
      "Default": "192.168.0.0/16"
    },
    "VSwitchCidrBlock": {
      "Type": "String",
      "Default": "192.168.0.0/24"
    }
  },
  "Metadata": {
    "ALIYUN::ROS::Interface": {
      "ParameterGroups": [
        {
          "Parameters": ["ZoneId"],
          "Label": {
            "default": "基础设置"
          }
        },
        {
          "Parameters": ["VpcCidrBlock", "VSwitchCidrBlock"],
          "Label": {
            "default": "资源设置"
          }
        }
      ]
    }
  }
}
```

使用模板示例代码创建资源栈时，您可以在资源编排控制台的基础设置和资源设置分组中集中设置参数。



7.更改集

7.1. 数据结构

本文介绍更改集的五种数据结构，包括Change、ResourceChange、ResourceChangeDetail、ResourceTargetDefinition和ResourcePropertyChange。

Change

参数	类型	描述
ResourceChange	结构体	更改的资源和操作。
Type	字符串	更改的实体类型。 取值： `Resource` ，表示资源。

ResourceChange

参数	类型	描述
Action	字符串	资源的相关操作。取值： • `Add` ：创建资源。 • `Modify` ：修改资源。 • `Remove` ：释放资源。
Details	数组	资源的修改详情。当 `Action` 取值为 `Modify` 时，该参数有效。 **说明** 仅ROS类型资源栈支持该参数。
LogicalResourceId	字符串	资源的逻辑ID，即模板中定义的资源名称。
PhysicalResourceId	字符串	资源的物理ID。当 `Action` 取值为 `Modify` 或 `Remove` 时，该参数有效。

参数	类型	描述

Replacement	字符串	当 `Action` 取值为 `Modify` 时，表示是否通过创建新资源并删除旧资源来替换资源。 取值：
		• ROS类型资源栈
		  ○ 当 `RequiresRecreation` 取值为 `Always` 、 `Evaluation` 取值为 `Static` 时： `True` ，表示ROS会通过创建新资源并删除旧资源来替换资源。
		  ○ 当 `RequiresRecreation` 取值为 `Never` 、 `Evaluation` 取值为 `Static` 时： `False` ，表示ROS不会通过创建新资源并删除旧资源来替换资源。
		  ○ 当 `RequiresRecreation` 取值为 `Always` 、 `Evaluation` 取值为 `Dynamic` 时： `Conditional` ，表示ROS可能会通过创建新资源并删除旧资源来替换资源。
		  说明 如果 `RequiresRecreation` 有多个取值，则 `Replacement` 的值取决于影响最大的更改， `Always` 、 `Conditional` 、 `Never` 影响递减。
		• Terraform类型资源栈
		  ○ `True` ：Terraform会通过创建新资源并删除旧资源来替换资源。
		  ○ `False` ：Terraform不会通过创建新资源并删除旧资源来替换资源。

ResourceType	字符串	资源类型。
Scope	字符串数组	当 `Action` 取值为 `Modify` 时，触发更新的参数。 取值： • `Properties` ：Resource中的Properties参数将触发更新。 • `Metadata` ：Resource中的Metadata参数将触发更新。 • `DeletionPolicy` ：Resource中的DeletionPolicy参数将触发更新。 **说明** 仅ROS类型资源栈支持该参数。

参数	类型	描述
PropertyChanges	ResourcePropertyChange结构体数组	当 `Action` 取值为 `Modify` 时，更新前后资源属性发生的变化。 **说明** 仅Terraform类型资源栈支持该参数。

ResourceChangeDetail

参数	类型	描述
ChangeSource	字符串	触发更新的原因。取值： • `ResourceReference`：引用了其他资源的物理ID，引用的资源物理ID可能发生了变化。 • `ParameterReference`：模板中引用的参数可能发生了变化。 • `ResourceAttribute`：引用了其他资源的输出属性，引用资源的输出属性可能发生了变化。 • `DirectModification`：直接修改了模板。 • `Automatic`：如果嵌套资源栈（`ALIYUN::ROS::Stack`）未进行修改，则ROS会将 `ChangeSource` 设置为 `Automatic`，因为嵌套资源栈中指定的模板可能已经发生了更改。在您对父资源栈运行更新之前，ROS不会对嵌套资源栈模板进行更改。 • `System`：虽然某些资源的输入未发生变化，但是特定的条件或内部机制也会触发更新，例如：`ALIYUN::ROS::WaitConditionHandle`。
CausingEntity	字符串	`ChangeSource` 关联的对象。对应关系如下： • `ResourceReference`：资源名。 • `ParameterReference`：参数名。 • `ResourceAttribute`：**资源名.参数名**。 • `DirectModification`：null。 • `Automatic`：资源名。 • `System`：资源名。
Evaluation	字符串	ROS是否可以确定目标值，以及在执行更改集之前目标值是否将更新。取值： • `Static`：ROS可以确定目标值，且目标值将更新。 • `Dynamic`：ROS无法确定目标值。因为更新资源栈时，目标值取决于内部函数的结果，例如：`Ref` 或 `Fn::GetAtt`。
Target	结构体	触发更新的参数的具体信息。

ResourceTargetDefinition

参数	类型	描述
Attribute	字符串	触发更新的参数。取值： • `Properties`：Resource中的Properties参数将触发更新。 • `Metadata`：Resource中的Metadata参数将触发更新。 • `DeletionPolicy`：Resource中的DeletionPolicy参数将触发更新。
Name	字符串	当`Attribute`为`Properties`时，表示具体的属性名，其他情况为`null`。
RequiresRecreation	字符串	当`Attribute`为`Properties`时，表示对此属性的更改是否导致重新创建资源。取值： • `Never`：不会导致重新创建资源。 • `Conditionally`：可能导致重新创建资源。 • `Always`：会导致重新创建资源。

ResourcePropertyChange

参数	类型	描述
Name	字符串	属性名。
BeforeValue	由属性本身决定	更新前的取值。
AfterValue	由属性本身决定	更新后的取值。 **说明** • 如果为字符串，且取值为`<known_after_apply>`，则表示创建或更新后才能获取。 • 如果为字符串，且取值为`<sensitive>`，则表示为敏感数据。

7.2. 通过创建更改集创建资源栈

ROS支持通过控制台、API或ALIYUN CLI的方式通过创建更改集创建资源栈，在您执行更改集之前可对资源栈进行检查和修改。对于您新创建的资源栈，只有在执行更改集成功后，才能成为有效的资源栈。

通过控制台创建资源栈

您可以在资源编排控制台，使用现有资源创建资源栈。创建后通过执行更改集完成资源导入。

具体操作，请参见使用现有资源创建资源栈。

通过API创建资源栈

您可以使用CreateChangeSet API通过更改集创建资源栈，使用PreviewStack API预览资源栈的配置结果。

通过创建更改集方式创建的资源栈的状态处于核对中（REVIEW_IN_PROGRESS）。该状态下的资源栈，不能创建新更改集。如果您确认后发现资源栈配置不符合要求，则可以直接删除资源栈，并重新通过创建更改集创建新的资源栈。

通过ALIYUN CLI创建资源栈

您可以使用aliyun ros CreateChangeSet 命令通过创建更改集创建资源栈。

您需要指定更改集类型为 `CREATE` ，并指定资源栈名称、模板、参数和更改集名称。例如：指定资源栈名称为 `test-create-change-set` ，更改集名称为 `test-create-change-set` 。

```
aliyun ros CreateChangeSet --ChangeSetType CREATE --TemplateURL oss://ros-templates/test-change-set.json?RegionId=cn-hangzhou --StackName test-create-change-set --ChangeSetName test-create-change-set --Parameters.1.ParameterKey Count --Parameters.1.ParameterValue 1
```

7.3. 更新资源栈

7.3.1. 概览

当您需要更新正在运行的资源栈时，可以使用更改集功能。您可以预览更改对资源栈正在运行资源造成的影响，只有在您执行更改集时，ROS才会对您的资源栈进行更改，您可以使用ROS控制台、ALIYUN CLI或ROS API来创建和管理更改集。

使用限制

更改集的使用限制如下：

- 一个资源栈最多同时存在20个更改集。
- 更改集只显示资源栈变化，不显示资源栈是否成功更新。
- 更改集不检查是否将超出账户限制、是否将更新不支持更新的资源、是否权限不足而无法修改资源，所有这些都将导致资源栈更新失败。如果更新失败，ROS将尝试将您的资源回滚到原始状态。

更新资源栈流程

使用更改集更新资源栈的流程如下：

1. 通过为您待更新的资源栈提交更改来创建更改集。您可以提交修改后的资源栈模板或者修改后的输入参数值，ROS将您的资源栈与所提交更改进行对比，生成更改集。
2. 查看更改集了解将更改资源栈设置和资源。例如，您可以查看ROS将添加、修改或删除的资源。
3. （可选）如果您在执行更改前，需要其他更改，则请创建其他更改集。创建多个更改集可帮助您了解和评估不同的更改对您的资源造成的影响。您还可以删除其他更改集，以免误执行不应该应用的更改集。
4. 执行您需要应用到资源栈的更改集。

> 注意 在您执行更改集后，ROS将自动删除与资源栈关联的所有更改集，因为他们不再适用于更新后的资源栈。

5. 资源栈开始更新，待更新完成后，您可以查看资源栈更新结果。

相关功能

更改集的相关功能如下表所示。

功能	说明
创建更改集	当您需要为正在运行的资源栈创建更改集时，请通过本操作修改模板或模板参数。
查看更改集	创建更改集之后，您可以查看更改记录和JSON格式的更改详细列表。
执行更改集	当您创建更改集后，只有成功执行更改集，对资源栈的更新才能生效。
删除更改集	当您不需要新建的更改集时，请及时删除，以免误执行，导致资源栈更改。

7.3.2. 创建更改集

您可以通过更改集功能更新资源栈的模板及模板参数。本文为您介绍如何创建更改集。

前提条件

请确保您已创建资源栈，操作方法请参见创建资源栈。

使用限制

只有以下状态的资源栈支持创建更改集：

状态	说明
CREATE_COMPLETE	资源栈创建成功。
UPDATE_FAILED	资源栈更新失败。
UPDATE_COMPLETE	资源栈更新完成。
ROLLBACK_COMPLETE	资源栈回滚完成。
ROLLBACK_FAILED	资源栈回滚失败。
IMPORT_CREATE_COMPLETE	通过资源导入创建资源栈成功。
IMPORT_UPDATE_COMPLETE	通过资源导入更新资源栈成功。
IMPORT_UPDATE_FAILED	通过资源导入更新资源栈失败。
IMPORT_UPDATE_ROLLBACK_COMPLETE	通过资源导入更新资源栈失败，回滚成功。

状态	说明
IMPORT_UPDATE_ROLLBACK_FAILED	通过资源导入更新资源栈失败，回滚失败。
CHECK_FAILED	资源栈校验失败。
CHECK_COMPLETE	资源栈校验完成。

创建更改集（控制台）

1. 登录资源编排控制台。
2. 在左侧导航栏，单击**资源栈**。
3. 在页面左上角的地域下拉列表，选择资源栈的所在地域。
4. 在目标资源栈操作列，选择 > **创建更改集**。

   您也可以单击**资源栈名称**下面的资源栈ID，在资源栈管理页面，选择**更改集**页签，然后单击**创建更改集**。
5. 在**选择模板**页面，根据所需选择已有模板或者示例模板，单击**下一步**。
6. 在**配置模板参数**页面，配置**更改集名称**和模板参数，然后单击**下一步**。

   > **说明** 模板参数是从模板中解析而来，请您根据控制台提示输入参数信息。
7. 在**配置更改集**页面，配置**资源栈策略**、**失败时回滚**、**超时设置**、**RAM角色**和**是否启用替换更新**，然后单击**下一步**。
8. 在**检查并确认**页面，单击**创建更改集**。

通过阿里云CLI创建更改集

您可以借助阿里云命令行工具 CLI（Alibaba Cloud CLI），通过调用命令**aliyun ros CreateChangeSet**来创建更改集。

您需要指定更改集类型为 `CREATE` ，并指定资源栈名称、模板、参数和更改集名称。例如：为资源栈创建名为 `test-change-set` 的更改集，更改集使用当前资源栈模板（ `oss://ros-templates/test-change-set.json?RegionId=cn-hangzhou` ）。

```
aliyun ros CreateChangeSet --TemplateURL oss://ros-templates/test-change-set.json?RegionId=cn-hangzhou --StackId <stack_id> --ChangeSetName test-change-set --Parameters.1.ParameterKey Count --Parameters.1.ParameterValue 1
```

7.3.3. 查看更改集

创建更改集之后，您可以查看更改记录和JSON格式的更改详细列表。

前提条件

请确保您已创建更改集，操作方法请参见创建更改集。

查看更改集（控制台）

1. 登录资源编排控制台。
2. 在左侧导航栏选择**资源栈**。

3. 单击**资源栈名称**下面的资源栈ID，在资源栈管理页面，选择**更改集**，查看资源栈的更改集的列表。
4. 单击目标更改集名称。

   在更改集详情页面，您可以查看更改集的基本信息、更改记录、模板及JSON更改。
   更改记录和**JSON更改**为您展示模板变更内容。如果您需要对模板执行其他更改，则请创建其他更改集。

查看更改集（ALIYUN CLI）

1. 执行命令**aliyun ros ListChangeSets**查看更改集所属资源栈ID。

   样例如下：

```
aliyun ros ListChangeSets --StackId <stack_id> --RegionId <region_id>
```

   ROS返回更改集所属资源栈信息列表如下：

```
{
        "TotalCount": 1,
        "PageSize": 10,
        "RequestId": "A94A31B7-EC3A-4528-90D8-FA31FA4D13BB",
        "PageNumber": 1,
        "ChangeSets": [
                {
                        "Status": "CREATE_COMPLETE",
                        "ChangeSetId": "<change_set_id>",
                        "ExecutionStatus": "AVAILABLE",
                        "CreateTime": "2020-03-03T06:36:20",
                        "ChangeSetType": "UPDATE",
                        "RegionId": "cn-hangzhou",
                        "ChangeSetName": "test-change-set",
                        "StackName": "test-change-set",
                        "StackId": "<stack_id>"
                }
        ]
}
```

2. 执行命令**aliyun ros GetChangeSet**查看更改集ID。

   样例如下：

```
aliyun ros GetChangeSet --ChangeSetId <change_set_id> --RegionId <region_id>
```

   ROS返回更改集信息列表如下：

```
{
        "ExecutionStatus": "AVAILABLE"
```

```
        "ExecutionStatus": "AVAILABLE",
        "Parameters": [
                {
                        "ParameterValue": "<account_id>",
                        "ParameterKey": "ALIYUN::AccountId"
                },
                {
                        "ParameterValue": "None",
                        "ParameterKey": "ALIYUN::NoValue"
                },
                {
                        "ParameterValue": "cn-hangzhou",
                        "ParameterKey": "ALIYUN::Region"
                },
                {
                        "ParameterValue": "<stack_id>",
                        "ParameterKey": "ALIYUN::StackId"
                },
                {
                        "ParameterValue": "test-change-set",
                        "ParameterKey": "ALIYUN::StackName"
                },
                {
                        "ParameterValue": "<tenant_id>",
                        "ParameterKey": "ALIYUN::TenantId"
                },
                {
                        "ParameterValue": "1",
                        "ParameterKey": "Count"
                }
        ],
        "TimeoutInMinutes": 60,
        "Changes": [
                {
                        "Type": "Resource",
                        "ResourceChange": {
                                "LogicalResourceId": "WaitConditionHandle",
                                "Replacement": "False",
                                "PhysicalResourceId": "WaitConditionHandle",
                                "ResourceType": "ALIYUN::ROS::WaitConditionHandle",
                                "Action": "Modify",
                                "Details": [
                                        {
                                                "Evaluation": "Static",
                                                "Target": {
                                                        "Name": "Count",
                                                        "RequiresRecreation": "Never",
                                                        "Attribute": "Properties"
                                                },
                                                "CausingEntity": "Count",
                                                "ChangeSource": "ParameterReference"
                                        },
                                        {
                                                "Evaluation": "Dynamic",
                                                "Target": {
```

```
                                                "Name": "Count",
                                                "RequiresRecreation": "Never",
                                                "Attribute": "Properties"
                                            },
                                            "ChangeSource": "DirectModification"
                                        }
                                    ],
                                    "Scope": [
                                        "Properties"
                                    ]
                                }
                            }
                ],
                "ChangeSetId": "<change_set_id>",
                "StackId": "<stack_id>",
                "DisableRollback": false,
                "ChangeSetName": "test-change-set",
                "ChangeSetType": "UPDATE",
                "StackName": "test-change-set",
                "Status": "CREATE_COMPLETE",
                "CreateTime": "2020-03-03T06:36:20",
                "RegionId": "cn-hangzhou",
                "RequestId": "DB9B48C8-C22D-4009-A3B0-85FDF3D26D2D"
}
```


`Changes` 属性列出对资源的更改，配置方法请参见数据结构。

7.3.4. 执行更改集

当您创建更改集后，只有成功执行更改集，对资源栈的更新才能生效。

前提条件

请确保您已创建更改集，操作方法请参见创建更改集。

背景信息

> **注意** 您执行更改集之后，ROS将自动删除与资源栈关联的其他更改集，因为他们对于更新后的资源栈失效。如果资源栈更新失败，则您需要重新创建更改集。

如果您在有资源栈策略的资源栈上执行更改集，则ROS将在更新资源栈时强制执行策略。执行更改集时，您不能指定覆盖现有策略的临时资源栈策略。要更新受保护的资源，您必须更新资源栈策略或者直接更新资源栈。

执行更改集（控制台）

1. 登录资源编排控制台。
2. 在左侧导航栏选择**资源栈**。
3. 单击**资源栈名称**下面的资源栈ID。
4. 在资源栈管理页面，单击**更改集**。
5. 执行更改集。
   - 方法一：在**更改集**页签，找到需要执行的更改集，单击对应**操作**列的**执行**。

- 方法二：在**更改集**页签，找到需要执行的更改集，单击更改集名称进入更改集管理页面。单击**执行更改集**。

ROS立即开始更新资源栈。

> **说明** 执行更改集时，会使用创建更改集时指定的RAM角色，您可以在**资源栈信息**页签查看RAM角色。

执行更改集（ALIYUN CLI）

通过命令**aliyun ros ExecuteChangeSet**执行更改集。

指定您需要执行更改集的ID，例如：

```
aliyun ros ExecuteChangeSet --ChangeSetId <change_set_id> --RegionId <region_id>
```

执行本命令之后，ROS开始更新资源栈。如果您需要查看更新资源栈的进度，请使用执行命令**aliyun ros GetStack**。

7.3.5. 删除更改集

当您不需要更改集时，请及时删除，以免误执行，导致资源栈更改。除非您删除更改集，否则ROS将保留所有更改集，直到您更新资源栈为止。

前提条件

请确保您已创建更改集，操作方法请参见创建更改集。

删除更改集（控制台）

1. 登录资源编排控制台。
2. 在左侧导航栏选择**资源栈**。
3. 单击**资源栈名称**下面的资源栈ID，在资源栈管理页面，单击**更改集**。

4. 在**更改集**页签，找到需要删除的更改集，单击对应操作列的**删除**。

5. 在弹出的对话框中，单击**确定**。

删除更改集（ALIYUN CLI）

通过命令aliyun ros DeleteChangeSet删除更改集。

指定您要删除的更改集的ID，样例如下：

```
aliyun ros DeleteChangeSet --ChangeSetId <change_set_id> --RegionId <region_id>
```

8.偏差检测

8.1. 概览

使用偏差检测可以识别在资源编排（以下简称ROS）管理之外进行配置更改的资源栈资源。您可以采取纠正措施，使资源栈资源再次与其在资源栈模板中的定义同步。例如直接更新存在偏差的资源，以便它们与其模板定义一致，或者通过模板纠正功能直接纠正模板。解决偏差有助于确保资源配置的一致性。
您可以在ROS之外更改资源，使用创建资源的底层服务直接编辑资源。例如，您可以使用ECS控制台来更新作为ROS资源栈一部分而创建的服务器实例。在ROS之外进行的更改会使资源栈的更新或删除操作复杂化。使用偏差检测可以识别在ROS管理之外进行配置更改的资源栈资源。

通过偏差检测，您可以检测资源栈的实际配置是否与其预期配置不同或已经存在偏差。使用ROS可以在整个资源栈或资源栈内的单个资源上检测偏差，您可以检测属性、资源是否被删除。

- 如果某个资源的实际属性值与预期属性值不同，则认为该资源已经存在偏差。
- 如果资源栈的一个或多个资源已经存在偏差，则认为资源栈已经存在偏差。

为了确定资源是否已经存在偏差，ROS将预期的资源属性值（如资源栈模板中所定义）指定为模板参数的值，并将预期值与这些资源属性的实际值进行比较。

> **说明** 如果资源的一个或多个属性值被删除或修改，则认为该资源已经存在偏差。ROS将生成已经存在偏差的资源栈中每个资源的详细信息。

检测偏差时的注意事项

ROS仅在支持偏差检测的资源上检测偏差，不支持偏差检测的资源会显示 `NOT_CHECKED` 状态。详情请参考支持偏差检测和资源导入的资源类型。

您可以在具有以下状态的资源栈上执行偏差检测：

- `CREATE_COMPLETE`
- `UPDATE_COMPLETE`
- `ROLLBACK_COMPLETE`
- `ROLLBACK_FAILED`
- `CHECK_COMPLETE`

当在资源栈上检测偏差时，ROS不会在任何属于该资源栈的嵌套资源栈上检测偏差。您也可以直接在嵌套资源栈上启动偏差检测操作。

ROS仅通过资源栈模板或指定模板参数来确定属性值的偏差，这不包括资源属性的默认值。要使ROS跟踪资源属性以确定偏差，需要显示设置的属性值。

为了在资源栈上成功执行偏差检测，您必须具有以下权限：

- 支持包含在资源栈中的偏差检测的每个资源的读取权限。例如，如果资源栈包含 `ALIYUN::VPC::EIP` 资源，则您必须具有 `vpc:DescribeEipAddresses` 权限才能在资源栈上进行偏差检测。
- 如果需要对资源栈进行偏差检测，则需要具有 `ros:DetectStackDrift` 权限。
- 如果需要对资源进行偏差检测，则需要具有 `ros:DetectStackResourceDrift` 权限。

在某些情况下，ROS可能无法返回准确的偏差结果。您应该了解这些情况，以便正确解释您的偏差检测结果。

- 在某些情况下，属性数组中包含的对象将报告为偏差，而实际上它们是从负责该资源的底层服务提供给属性的默认值。

- 您可以在资源栈模板中指定某些资源属性。ROS无法将其与生成的资源栈资源中的属性进行比较，因此这些属性不能包括在偏差检测结果中。这些属性可分为四大类：
  - ROS无法在资源栈模板中映射回初始资源属性值的属性值。
  - 负责资源的服务不返回的属性值。
  - 某些属性值被有意设计成永远不会由资源所属的服务返回。这些属性值往往包含机密信息，例如密码或其他不应泄露的敏感数据。
  - ROS尚未支持的资源属性。

  资源的属性究竟是否支持偏差检查，可以通过GetResourceType接口查询。以查询 `ALIYUN::ESS::ScalingRule` 资源的返回值为例，最外层的 `SupportDriftDetection` 字段表明资源是否支持偏差检查。如果取值为 `true` ，表明资源支持偏差检查，此时每个属性都有 `SupportDriftDetection` 字段表明该属性是否支持偏差检查。

```
{
    ...
    "ResourceType": "ALIYUN::ESS::ScalingRule",
    "Properties": {
        "ScalingRuleName": {
            ...
            "SupportDriftDetection": true
        },
        ...
    },
    "SupportDriftDetection": true
}
```

偏差检测状态码

以下给出了资源偏差状态的具体解释。

- 偏差检测操作状态：描述偏差检测操作的当前状态。
- 偏差状态。
  - 资源栈组：根据属于资源栈组的资源栈实例的偏差状态来描述资源栈组的总体偏差状态。
  - 资源栈实例：根据资源栈实例关联资源栈的偏差状态来描述资源栈实例的偏差状态。
  - 资源栈：根据资源栈资源的偏差状态来描述资源栈的总体偏差状态。
- 资源偏差状态：描述单个资源的偏差状态。

ROS分配给资源栈偏差检测操作的状态码如下表所示。

偏差检测操作状态	描述
`DETECTION_COMPLETE`	已经对支持偏差检测的资源栈中的所有资源成功完成了资源栈偏差检测操作。
`DETECTION_FAILED`	资源栈偏差检测操作对于资源栈中的至少一个资源检测失败。
`DETECTION_IN_PROGRESS`	资源栈偏差检测操作目前正在进行中。

ROS分配给资源栈的偏差状态码如下表所示。

偏差状态	描述
DRIFTED	• 资源栈：资源栈与其预期模板配置不同或已经存在偏差。如果资源栈的一个或多个资源已经存在偏差，则认为资源栈已经存在偏差。 • 资源栈实例：如果与资源栈实例关联的资源栈已经存在偏差，则认为该实例已经存在偏差。 • 资源栈组：如果一个或多个资源栈实例已经存在偏差，则认为资源栈组已经存在偏差。
NOT_CHECKED	ROS尚未检查资源栈、资源栈组或资源栈实例是否与其预期模板配置不同。
IN_SYNC	偏差检测支持资源的当前配置与其预期的模板配置相匹配。 **说明** 资源栈、资源栈组或资源栈实例没有支持偏差检测的资源时，也具有 IN_SYNC 状态。

ROS分配给资源栈资源的偏差代码如下表所示。

资源偏差状态	描述
DELETED	资源与预期的模板配置不同，因为资源已被删除。
MODIFIED	资源与预期的模板配置不同。
NOT_CHECKED	ROS没有检查资源是否与预期的模板配置不同。
IN_SYNC	资源的当前配置与其预期的模板配置相匹配。

ROS分配给与预期模板配置不同的资源属性的差异类型状态代码如下表所示。

属性差异类型	描述
ADD	已将值添加到数据类型为数组或列表的资源属性。
REMOVE	属性已从当前资源配置中删除。
NOT_EQUAL	当前属性值与资源栈模板中定义的预期值不同。

相关内容

偏差检测的相关内容如下表所示。

内容	描述
检测资源栈的偏差状态	在资源栈上执行偏差检测操作，可以确定资源栈是否已偏离其预期的模板配置，并返回有关支持偏差检测的资源栈中每个资源的偏差状态详情。

内容	描述
检测资源的偏差状态	对资源栈上单个资源执行偏差检测操作，可以确定资源是否已偏离其预期的模板配置。
支持偏差检测和资源导入的资源类型	ROS仅在支持偏差检测的资源上检测偏差。
检测资源栈组的偏差状态	对资源栈组执行偏差检测，可以确定属于该资源栈组的任何资源栈实例是否与它们的预期配置不同或已经存在偏差。
纠正资源栈的偏差状态	纠正资源栈偏差有助于确保资源配置的一致性，使资源栈资源与资源栈模板中定义的资源保持同步。

8.2. 检测资源栈的偏差状态

通过偏差检测，您可以检测资源栈的实际配置是否与其模版配置不同或已经偏差。

前提条件

请确保您已经创建了资源栈。具体操作，请参见创建资源栈。

使用限制

- 只有以下状态的资源栈支持偏差检测。

状态	说明
CREATE_COMPLETE	资源栈创建成功。
UPDATE_FAILED	资源栈更新失败。
UPDATE_COMPLETE	资源栈更新完成。
ROLLBACK_COMPLETE	资源栈回滚完成。
ROLLBACK_FAILED	资源栈回滚失败。
CHECK_COMPLETE	资源栈校验完成。
CHECK_FAILED	资源栈校验失败。

- 目前只支持部分资源的偏差检测。更多信息，请参见支持偏差检测和资源导入的资源类型。

检测偏差（控制台）

1. 登录资源编排控制台。
2. 在左侧导航栏，单击**资源栈**。
3. 在页面左上角的地域下拉列表，选择资源栈的所在地域。
4. 在目标资源栈的操作列，选择 > **检测偏差**。

   您也可以单击**资源栈名称**下面的资源栈ID，在**资源栈信息**页签，单击**检测偏差**。
5. 在**偏差**页签，查看资源栈的**偏差状态**、**上一次偏差检查时间**和**资源偏差状态**。

检测偏差（阿里云CLI）

您可以借助阿里云命令行工具 CLI（Alibaba Cloud CLI），调用偏差检测相关接口检测资源栈偏差。具体如下：

- 调用DetectStackDrift接口对资源栈进行偏差检测。您需要指定资源栈ID，以及用于此次偏差检测操作筛选条件的特定资源名称。
  输入以下命令：

  ```
  aliyun ros DetectStackDrift --StackId bc1a154f-d073-4e77-9ae5-323d3b23****
  ```

  预期输出：

  ```
  {
      "DriftDetectionId": "ad9cf0c7-938e-40b3-9466-ec9f25a1****",
      "RequestId": "B288A0BE-D927-4888-B0F7-B35EF84B6E6F"
  }
  ```

- 调用GetStackDriftDetectionStatus接口查询偏差检测的状态。此接口将获取 `DetectStackDrift` 返回的资源栈偏差检测ID。
  在以下示例中，我们采用了如上 `DetectStackDrift` 示例返回的资源栈偏差检测ID，并将其作为参数传递给 `GetStackDriftDetectionStatus` 。此参数返回操作详细信息，显示偏差检测操作已完成。
  输入以下命令：

  ```
  aliyun ros GetStackDriftDetectionStatus --DriftDetectionId ad9cf0c7-938e-40b3-9466-ec9f25
  a1****
  ```

  预期输出：

  ```
  {
      "RequestId": "52398D3A-E868-4F95-8B5E-6A2DFB778B16",
      "DriftDetectionTime": "2020-03-17T07:21:17",
      "DriftDetectionStatusReason": "Detect stack drift successfully",
      "DriftedStackResourceCount": 2,
      "DriftDetectionStatus": "DETECTION_COMPLETE",
      "StackDriftStatus": "DRIFTED",
      "DriftDetectionId": "ad9cf0c7-938e-40b3-9466-ec9f25a1****",
      "StackId": "bc1a154f-d073-4e77-9ae5-323d3b23****"
  }
  ```

- 调用ListStackResourceDrifts接口查询资源栈的资源偏差详情。
  输入以下命令：

  ```
  aliyun ros ListStackResourceDrifts --StackId bc1a154f-d073-4e77-9ae5-323d3b23****
  ```

  预期输出：

```
{
  "ResourceDrifts": [
    {
      "ResourceDriftStatus": "MODIFIED",
      "LogicalResourceId": "Vpc1",
      "PropertyDifferences": [
        {
          "ActualValue": "test11",
          "PropertyPath": "/Description",
          "ExpectedValue": "test1",
          "DifferenceType": "NOT_EQUAL"
        }
      ],
      "PhysicalResourceId": "vpc-m5euqfvmzygb7xqmx****",
      "ExpectedProperties": "{\"CidrBlock\": \"192.168.0.0/16\", \"Description\": \"test1
\", \"VpcName\": \"test1\"}",
      "DriftDetectionTime": "2020-03-17T07:21:17",
      "ResourceType": "ALIYUN::ECS::VPC",
      "ActualProperties": "{\"CidrBlock\": \"192.168.0.0/16\", \"Description\": \"test11\
", \"VpcName\": \"test1\"}",
      "StackId": "bc1a154f-d073-4e77-9ae5-323d3b23****"
    },
    {
      "ResourceDriftStatus": "DELETED",
      "LogicalResourceId": "Vpc2",
      "PhysicalResourceId": "vpc-m5exf3skxrxtvtkbc****",
      "DriftDetectionTime": "2020-03-17T07:21:17",
      "ResourceType": "ALIYUN::ECS::VPC",
      "StackId": "bc1a154f-d073-4e77-9ae5-323d3b23****"
    }
  ],
  "RequestId": "8E1DE57B-6124-482B-8283-EF5562653308"
}
```


8.3. 检测资源的偏差状态

您可以对资源栈上单个资源执行偏差检测操作，以便确定资源是否已偏离其预期的模板配置。

前提条件

请确保您已经在整个资源栈上进行了偏差检测，操作方法请参见检测资源栈的偏差状态。

检测偏差（控制台）

1. 登录ROS控制台。
2. 在左侧导航栏，单击**资源栈**。
3. 在**资源栈列表**页面，单击目标资源栈ID。
4. 单击**偏差**页签。
5. 在**资源偏差状态**区域，单击资源名称右侧的**检测资源偏差**，检测单个资源的偏差状态。
6. 在**资源偏差状态**区域，单击资源名称右侧的**查看偏差详细信息**，查看资源物理ID、偏差状态、资源类型以及上一次偏差检查时间。

检测偏差（阿里云CLI）

您可以借助阿里云命令行工具 CLI（Alibaba Cloud CLI），调用DetectStackResourceDrift接口对资源进行偏差检测。您需要指定资源栈ID、资源名称以及资源栈所属的地域ID。

输入以下命令：

```
aliyun ros DetectStackResourceDrift --StackId a4dffae5-d2a5-4188-b8b5-69dbe8a2**** --Logica
lResourceId Vpc --RegionId cn-beijing
```

预期输出：

```
{
  "LogicalResourceId": "Vpc",
  "ResourceDriftStatus": "IN_SYNC",
  "RequestId": "7D9AAA7E-F165-5EB8-AFFA-B95A572B9921",
  "PhysicalResourceId": "vpc-2zevdu0ktm8tmzry9****",
  "ExpectedProperties": "{\"CidrBlock\": \"192.168.0.0/16\", \"VpcName\": \"bff\"}",
  "DriftDetectionTime": "2021-10-20T06:13:01",
  "ResourceType": "ALIYUN::ECS::VPC",
  "ActualProperties": "{\"CidrBlock\": \"192.168.0.0/16\", \"VpcName\": \"bff\"}",
  "StackId": "a4dffae5-d2a5-4188-b8b5-69dbe8a2****"
}
```

8.4. 支持偏差检测和资源导入的资源类型

本文介绍了支持偏差检测和资源导入的资源类型。

云服务	资源类型
云服务器ECS	• ALIYUN::ECS::VPC • ALIYUN::ECS::VSwitch • ALIYUN::ECS::SecurityGroup • ALIYUN::ECS::Instance • ALIYUN::ECS::NetworkInterface • ALIYUN::ECS::NetworkInterfaceAttachment • ALIYUN::ECS::SecurityGroupIngress • ALIYUN::ECS::Disk • ALIYUN::ECS::DiskAttachment • ALIYUN::ECS::SecurityGroupEgress • ALIYUN::ECS::Snapshot
弹性伸缩	• ALIYUN::ESS::ScalingRule • ALIYUN::ESS::ScalingConfiguration • ALIYUN::ESS::ScalingGroup • ALIYUN::ESS::ScalingGroupEnable

云服务	资源类型
访问控制RAM	• ALIYUN::RAM::Role • ALIYUN::RAM::AttachPolicyToRole • ALIYUN::RAM::ManagedPolicy
负载均衡SLB	• ALIYUN::SLB::Listener • ALIYUN::SLB::LoadBalancer • ALIYUN::SLB::VServerGroup • ALIYUN::SLB::BackendServerAttachment • ALIYUN::SLB::MasterSlaveServerGroup
专有网络VPC	• ALIYUN::VPC::EIP • ALIYUN::VPC::EIPAssociation • ALIYUN::VPC::CommonBandwidthPackageIp • ALIYUN::VPC::SnatEntry • ALIYUN::VPC::CommonBandwidthPackage • ALIYUN::VPC::NatGateway
云解析DNS	ALIYUN::DNS::DomainRecord
文件存储NAS	ALIYUN::NAS::FileSystem
云数据库Redis版	• ALIYUN::REDIS::Instance • ALIYUN::REDIS::Whitelist
云数据库MongoDB版	ALIYUN::MONGODB::Instance
日志服务SLS	• ALIYUN::SLS::Project • ALIYUN::SLS::Logstore • ALIYUN::SLS::Index • ALIYUN::SLS::Alert • ALIYUN::SLS::Savedsearch
函数计算FC	• ALIYUN::FC::Function • ALIYUN::FC::Service • ALIYUN::FC::Trigger • ALIYUN::FC::Version • ALIYUN::FC::CustomDomain • ALIYUN::FC::Alias

云服务	资源类型
API网关	• ALIYUN::ApiGateway::Deployment • ALIYUN::ApiGateway::Api • ALIYUN::ApiGateway::Group • ALIYUN::ApiGateway::Authorization • ALIYUN::ApiGateway::App
表格存储	• ALIYUN::OTS::Instance • ALIYUN::OTS::Table • ALIYUN::OTS::VpcBinder
对象存储OSS	ALIYUN::OSS::Bucket
阿里云关系型数据库RDS	ALIYUN::RDS::DBInstance

8.5. 检测资源栈组的偏差状态

通过对资源栈组执行偏差检测，您可以确定属于该资源栈组的任何资源栈实例是否与它们的预期配置不同或已经存在偏差。

前提条件

请确保您已经创建资了源栈组。具体操作，请参见步骤二：创建资源栈组。

背景信息

ROS对资源栈组执行偏差检测时，会对与资源栈组中的每个资源栈实例关联的资源栈执行偏差检测。ROS将资源栈中的每个资源的当前状态与该资源的预期状态（通过资源栈的模板以及指定输入参数定义）进行比较，如果资源的当前状态不同于预期状态，则认为该资源已经存在偏差。

- 如果资源栈中的一个或多个资源已经存在偏差，则将资源栈本身视为已经存在偏差，并将与资源栈关联的资源栈实例也视为已经存在偏差。
- 如果资源栈组中的一个或多个资源栈实例已经存在偏差，则将资源栈组本身视为已经存在偏差。

偏差检测判定标准是在ROS外部对资源栈进行的更改，直接通过ROS对资源栈进行的更改（而不是在资源栈组级别）不会被视为存在偏差。例如：假设您具有一个与资源栈组的资源栈实例关联的资源栈。如果您使用ROS更新该资源栈来使用不同模板，则不会被视为偏差，即使该资源栈现在具有与属于资源栈组的其他资源栈不同的模板。这是因为，该资源栈仍与它在ROS中的预期模板和参数配置相匹配。

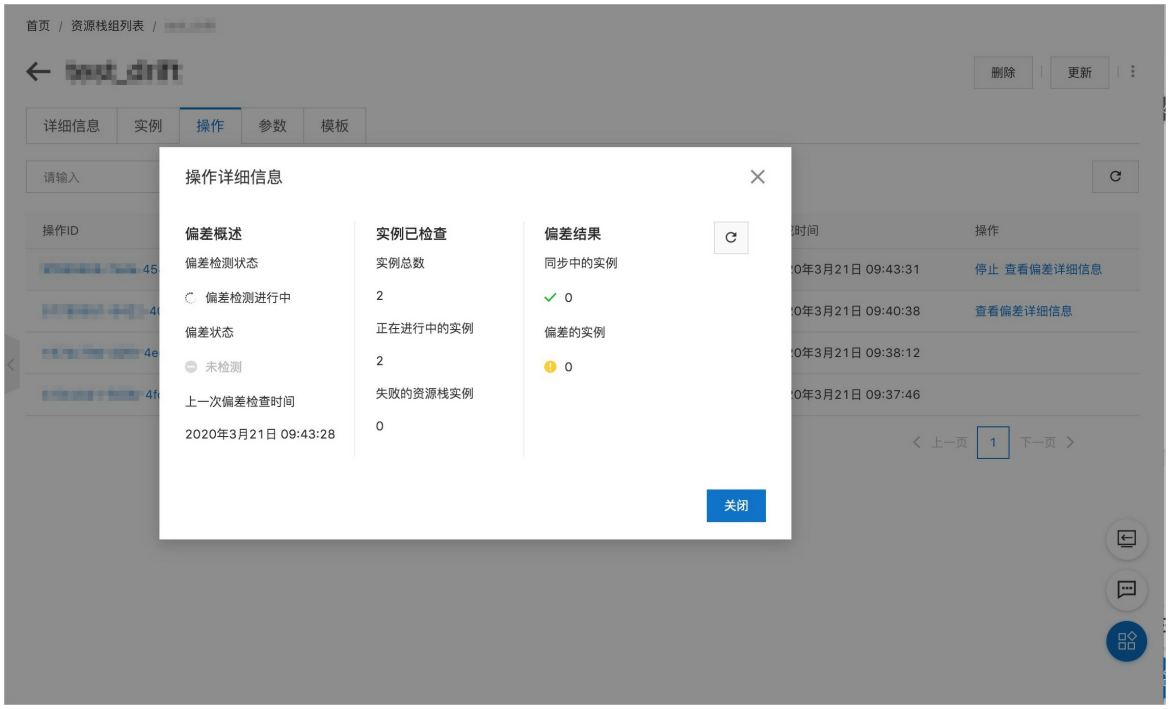
在资源栈组偏差检测时，ROS会分别对资源栈组中的每个资源栈执行偏差检测，它在确定资源栈是否偏离时考虑所有覆盖的参数值。如果直接对与资源栈实例关联的资源栈执行偏差检测，则无法从**资源栈组**页面中查看这些偏差结果。

检测偏差（控制台）

1. 登录ROS控制台。
2. 在左侧导航栏，单击**资源栈组**。
3. 在**资源栈组列表**页面，单击资源栈组名称。
4. 在**详细信息**页签，单击 ⋮ 图标，然后单击**检测偏差**。

说明 ROS会弹出提示窗口，说明已经为选定的资源栈组启动偏差检测。

5. 在弹出的**检测偏差**对话框，填写**最大并发账户数**和**容错**信息，单击**确认**。
6. （可选）单击**操作**页签，找到偏差检测操作，单击右侧**操作**列的**查看偏差详细信息**，可以监控偏差检测操作进度。



说明

- 您每次只能对给定资源栈组运行一次偏差检测操作。即使您关闭信息窗口，ROS也会继续执行偏差检测操作。
- 偏差检测操作可能需要几分钟，具体时间取决于资源栈组中包含的资源栈实例数量以及资源栈组中包含的资源数量。

7. 单击**实例**页签，查看偏差检测结果。

> 说明
>
> - 您可以在**资源栈ID**列查看与每个资源栈实例关联的资源栈ID，在**偏差状态**列查看该资源栈的偏差状态。如果资源栈的一个或多个资源已偏离，则认为资源栈已偏离。
> - 要查看与特定资源栈实例关联的资源栈的偏差检测结果，您可以记录资源栈实例的阿里云账号、资源栈名称和地域，登录包含资源栈实例的阿里云账号查看偏差结果。更多信息，请参见检测资源栈的偏差状态。

检测偏差（阿里云CLI）

使用 `aliyun ros` 在资源栈组上检测偏差。

命令	描述
`DetectStackGroupDrift`	对资源栈组启动偏差检测操作。
`GetStackGroupOperation`	查询资源栈组偏差检测操作的状态。
`StopStackGroupOperation`	停止资源栈组的偏差检查操作。

在偏差检测操作完成后，可以通过以下命令返回所需的偏差信息：

- 使用 `GetStackGroup` 返回有关资源栈组的详细信息，包括有关资源栈组上次完成的偏差操作的详细信息（不包括有关正在执行的偏差操作的信息）。
- 使用 `ListStackInstances` 返回属于资源栈组的资源栈实例列表，包括每个实例的偏差状态和上次偏差检查时间。
- 使用 `GetStackInstance` 返回有关特定资源栈实例的详细信息，包括其偏差状态和上次偏差检查时间。

8.6. 纠正资源栈的偏差状态

纠正资源栈偏差有助于确保资源配置的一致性，使资源栈资源与资源栈模板中定义的资源保持同步。

背景信息

纠正资源栈偏差，一般采用如下两个方法：

- 更新资源栈：通过更新资源栈，使资源与模板保持同步。详情请参见更新资源栈。
- 纠正模板：通过纠正模板，使资源与模板保持同步。本文重点介绍该方法。

纠正模板（控制台）

1. 登录ROS控制台。
2. 在左侧导航栏单击**资源栈**。
3. 在**资源栈列表**中，单击资源栈名称下的ID。
4. 单击**偏差**页签。
5. 在**偏差**页面，单击**检测资源栈偏差**检测偏差。
6. 在**资源栈偏差状态**区域，单击**纠正**。

说明 只有处于偏差状态的资源栈才会显示**纠正**。

7. 在**偏差纠正**页面，选择要纠正的资源。

8. 在**偏差纠正**页面，单击**预览**，对比纠正前后的模板内容。

   左侧为纠正前，右侧为纠正后。

9. 单击**确认**执行纠正。

   纠正完成后，重新进行偏差检测，查看检测结果。

纠正模板（阿里云CLI）

使用 `aliyun ros UpdateStackTemplateByResources` 对存在偏差的资源栈模板进行纠正，参数与UpdateStackTemplateByResources接口相同，详情请参见UpdateStackTemplateByResources。

```
$ aliyun ros UpdateStackTemplateByResources --StackId 4334b961-3bfd-419e-9a00-23a95e******
{
    "RequestId": "B288A0BE-D927-4888-B0F7-B35EF84B6E6F",
    "NewTemplateBody": "{\"ROSTemplateFormatVersion\": \"2015-09-01\", \"Resources\": {\"Vp
c\": {\"Type\": \"ALIYUN::ECS::VPC\", \"Properties\": {\"VpcName\": \"test\", \"CidrBlock\"
: \"192.168.0.0/16\", \"Description\": \"test2\"}}}, \"Outputs\": {\"VpcId\": {\"Value\": {
\"Fn::GetAtt\": [\"Vpc\", \"VpcId\"]}}}}",
    "OldTemplateBody": "{\"ROSTemplateFormatVersion\": \"2015-09-01\", \"Resources\": {\"Vp
c\": {\"Type\": \"ALIYUN::ECS::VPC\", \"Properties\": {\"VpcName\": \"test\", \"CidrBlock\"
: \"192.168.0.0/16\", \"Description\": \"test1\"}}}, \"Outputs\": {\"VpcId\": {\"Value\": {
\"Fn::GetAtt\": [\"Vpc\", \"VpcId\"]}}}}"
}
```

8.7. 偏差检测状态码

本文为您介绍资源栈偏差检测操作状态码、资源偏差状态码、资源属性差异类型状态码以及资源栈、资源栈实例和资源栈组偏差状态码。

资源栈偏差检测操作状态码

资源栈偏差检测操作状态码用于描述偏差检测操作的状态。

状态码	说明
`DETECTION_COMPLETE`	已经对支持偏差检测的资源栈中的所有资源成功完成了资源栈偏差检测操作。
`DETECTION_FAILED`	资源栈偏差检测操作对于资源栈中的至少一个资源检测失败。

状态码	说明
DETECTION_IN_PROGRESS	资源栈偏差检测操作目前正在进行中。

资源偏差状态码

资源偏差状态码用于描述单个资源的偏差状态。

状态码	说明
DELETED	资源与预期的模板配置不匹配，因为资源已被删除。
MODIFIED	资源与预期的模板配置不匹配。
NOT_CHECKED	ROS没有检查资源是否与预期的模板配置不匹配。
IN_SYNC	资源的当前配置与其预期的模板配置相匹配。

资源属性差异类型状态码

资源属性差异类型状态码用于资源属性的偏差状态。

状态码	描述
ADD	已将值添加到数据类型为数组或列表的资源属性。
REMOVE	属性已从当前资源配置中删除。
NOT_EQUAL	当前属性值与资源栈模板中定义的预期值不匹配。

资源栈、资源栈实例和资源栈组偏差状态码

资源栈、资源栈实例和资源栈组偏差状态码用于描述资源栈、资源栈实例和资源栈组的偏差状态。

状态码	说明
DRIFTED	• 资源栈：资源栈与其预期模板配置不匹配或已经存在偏差。如果资源栈的一个或多个资源已经存在偏差，则认为资源栈已经存在偏差。 • 资源栈实例：如果与资源栈实例关联的资源栈已经存在偏差，则认为该实例已经存在偏差。 • 资源栈组：如果一个或多个资源栈实例已经存在偏差，则认为资源栈组已经存在偏差。
NOT_CHECKED	ROS未检查资源栈、资源栈组或资源栈实例的偏差状态。
IN_SYNC	偏差检测支持资源的当前配置与其预期的模板配置相匹配。 **说明** 资源栈、资源栈组或资源栈实例没有支持偏差检测的资源时，也具有 IN_SYNC 状态。

9.Terraform

9.1. 概览

Terraform是一个开源的自动化资源编排工具。资源编排服务ROS（Resource Orchestration Service）为Terraform提供了托管的能力，您可以创建Terraform类型的模板和资源栈，编排阿里云、AWS或Azure的资源。Terraform功能兼容ROS API，您只需创建Terraform类型模板，即可调用ROS API实现相应功能。Terraform相关内容如下表所示：

内容	说明
创建Terraform类型资源栈	创建Terraform类型资源栈，您需要编写Terraform类型模板、配置模板参数，并配置超时设置、删除保护等内容。
创建Terraform类型模板	创建Terraform类型模板，定义阿里云、AWS或Azure资源、参数以及资源间的依赖关系。
Terraform类型模板结构	介绍Terraform类型模板结构。
Terraform支持的功能和资源	介绍Terraform支持的控制台功能、API和资源。

9.2. 创建Terraform类型资源栈

资源编排服务ROS（Resource Orchestration Service）为Terraform提供了托管的能力，您可以创建Terraform类型的资源栈编排阿里云、AWS或Azure的资源。

背景信息

关于Terraform类型模板结构的详情，请参见Terraform类型模板结构。

操作步骤

1. 登录资源编排控制台。
2. 在左侧导航栏，单击**资源栈**。
3. 在页面左上角的地域下拉列表，选择资源栈的所在地域。
4. 在**资源栈列表**页面，单击**创建资源栈**，然后在下拉列表中选择**使用新资源（标准）**。
5. 在**选择模板**页面，在**指定模板**区域选择**选择已有模板**。
6. 选择**模板录入方式**为**输入模板**，并选择**模板内容**为**Terraform**。
7. 编写Terraform类型模板，单击**下一步**。

   以创建一个专有网络（VPC）下的交换机（vSwitch）为例，介绍Terraform类型模板编写方法。

   i. 创建 `modules/vpc/main.tf` 文件，编辑内容，创建一个VPC。

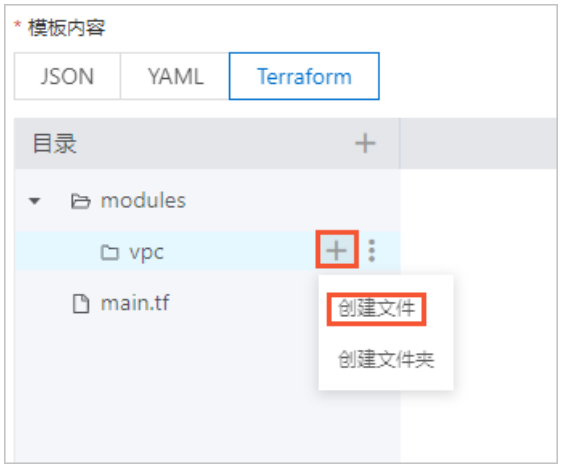
a. 单击目录右侧+，然后单击**创建文件夹**。

b. 在弹出的**创建文件夹**对话框中，输入modules，在目录下创建名为modules的文件夹。

c. 鼠标悬停在modules文件夹，单击右侧+，然后单击**创建文件夹**。

d. 在弹出的**创建文件夹**对话框中，输入 `vpc` ，在modules文件夹下创建名为 `vpc` 的文件夹。

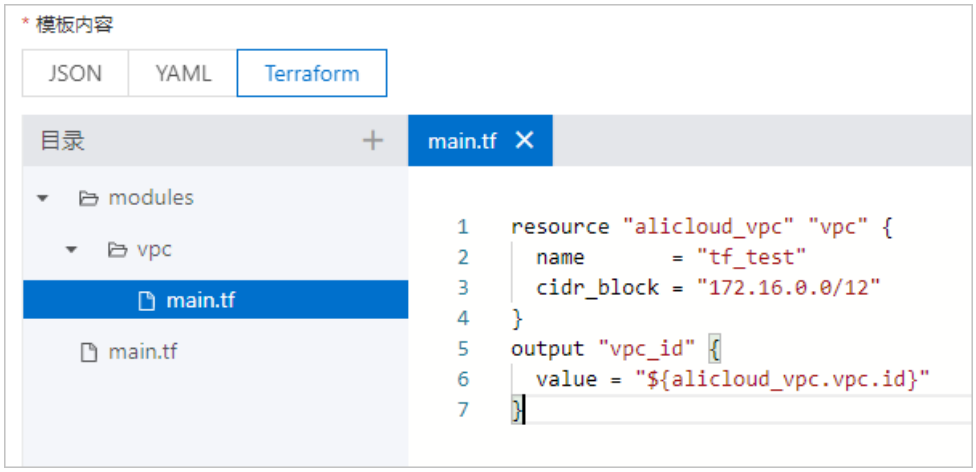
e. 鼠标悬停在 `vpc` 文件夹，单击右侧+，然后单击**创建文件**。



f. 在弹出的**创建文件**对话框中，输入main.tf，在 `vpc` 文件夹下创建main.tf文件。

g. 单击main.tf，在右侧编辑框输入如下代码，创建一个VPC。

```
resource "alicloud_vpc" "vpc" {
  name       = "tf_test"
  cidr_block = "172.16.0.0/12"
}
output "vpc_id" {
  value = "${alicloud_vpc.vpc.id}"
}
```



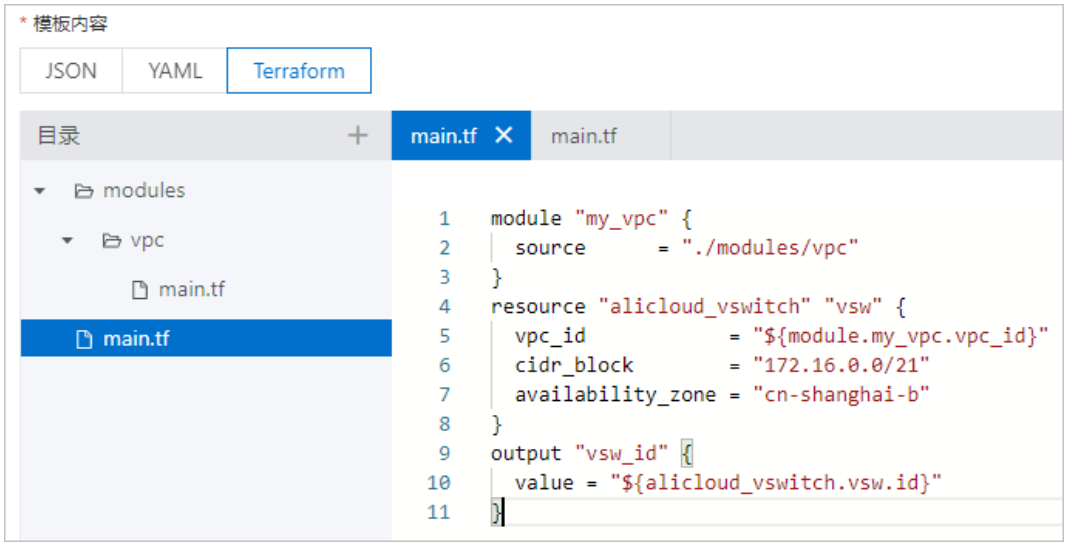
ii. 编辑根目录下的main.tf文件，创建一个专有网络（VPC）下的交换机（vSwitch）。

a. 单击根目录下的main.tf文件。

b. 在右侧编辑框输入如下代码，创建一个vSwitch。

```
module "my_vpc" {
  source       = "./modules/vpc"
}
resource "alicloud_vswitch" "vsw" {
  vpc_id            = "${module.my_vpc.vpc_id}"
  cidr_block        = "172.16.0.0/21"
  availability_zone = "cn-shanghai-b"
}
output "vsw_id" {
  value = "${alicloud_vswitch.vsw.id}"
}
```

说明 模板中可用区 `availability_zone` 需要在资源栈所属地域中。



8. 在**配置模板参数**页面，配置**资源栈名称**，单击**下一步**。
9. 在**配置资源栈**页面，配置**超时设置**、**删除保护**和**标签**，单击**下一步**。
10. 在**检查并确认**页面，单击**创建**。

说明 资源栈创建成功后，您可以单击**资源栈**，在**资源栈列表**单击资源栈ID进入资源栈详情页，查看基本信息、事件、资源、输出、模板等信息。

9.3. 创建Terraform类型模板

资源编排服务ROS（Resource Orchestration Service）为Terraform提供了托管的能力，您可以创建Terraform类型的模板，定义阿里云、AWS或Azure资源，配置资源参数和资源间的依赖关系。

背景信息

关于Terraform类型模板结构的详情，请参见Terraform类型模板结构。

操作步骤

1. 登录资源编排控制台。

2. 在左侧导航栏，选择**模板>我的模板**。
3. 在**我的模板**页面，单击**创建模板**。
4. 在弹出的**创建模板**对话框，填写**模板名称**和**模板描述**。
5. 选择**模板录入方式**为**输入模板**，选择**模板内容**为**Terraform**。
6. 编写Terraform类型模板。

   以创建一个专有网络（VPC）下的交换机（vSwitch）为例，介绍Terraform类型模板编写方法。

   i. 创建 `modules/vpc/main.tf` 文件，编辑内容，创建一个VPC。

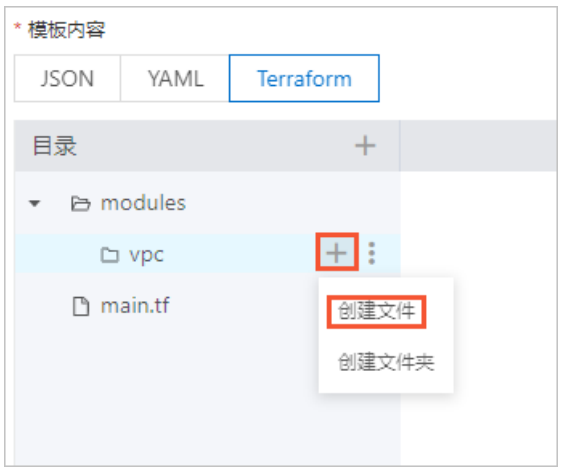
      a. 单击目录右侧**+**，然后单击**创建文件夹**。

      b. 在弹出的**创建文件夹**对话框中，输入modules，在目录下创建名为modules的文件夹。

      c. 鼠标悬停在modules文件夹，单击右侧**+**，然后单击**创建文件夹**。

      d. 在弹出的**创建文件夹**对话框中，输入 `vpc` ，在modules文件夹下创建名为 `vpc` 的文件夹。

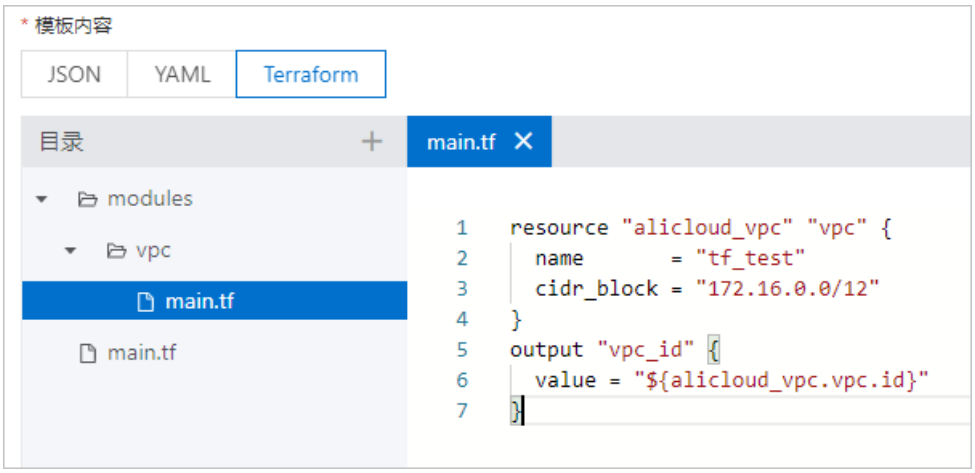
      e. 鼠标悬停在 `vpc` 文件夹，单击右侧**+**，然后单击**创建文件**。



f. 在弹出的**创建文件**对话框中，输入main.tf，在 `vpc` 文件夹下创建main.tf文件。

g. 单击main.tf，在右侧编辑框输入如下代码，创建一个VPC。

```
resource "alicloud_vpc" "vpc" {
  name       = "tf_test"
  cidr_block = "172.16.0.0/12"
}
output "vpc_id" {
  value = "${alicloud_vpc.vpc.id}"
}
```

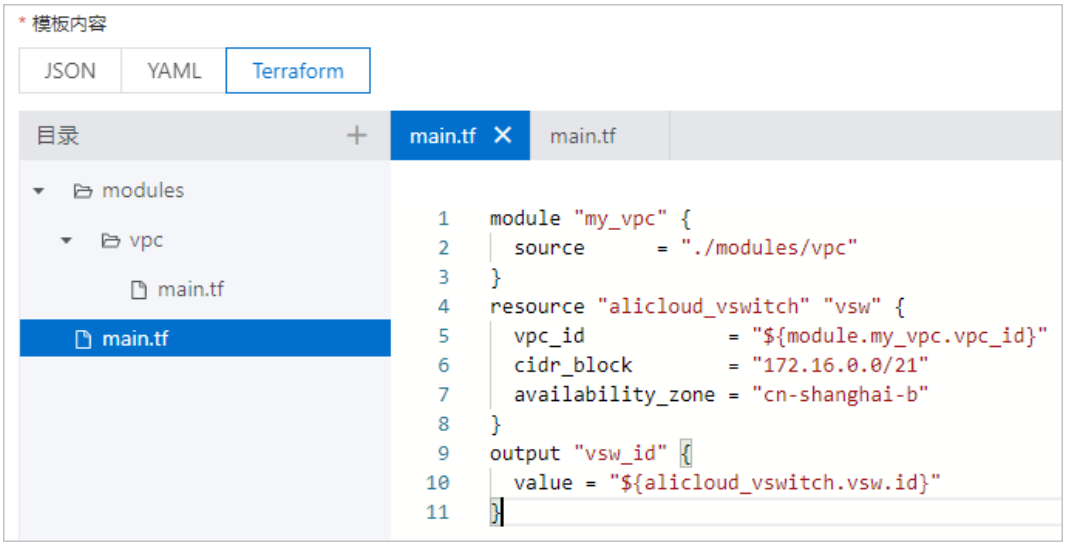


ii. 编辑根目录下的main.tf文件，创建一个专有网络（VPC）下的交换机（vSwitch）。

a. 单击根目录下的main.tf文件。

b. 在右侧编辑框输入如下代码，创建一个vSwitch。

```
module "my_vpc" {
  source      = "./modules/vpc"
}
resource "alicloud_vswitch" "vsw" {
  vpc_id            = "${module.my_vpc.vpc_id}"
  cidr_block        = "172.16.0.0/21"
  availability_zone = "cn-shanghai-b"
}
output "vsw_id" {
  value = "${alicloud_vswitch.vsw.id}"
}
```



7. 单击**确定**。

9.4. Terraform类型模板结构

Terraform类型模板是资源编排服务ROS（Resource Orchestration Service）托管Terraform后支持的模板，用于编排阿里云、AWS或Azure的资源。您可以在模板中定义资源、参数以及资源间的依赖关系。

模板结构

Terraform类型模板由7部分组成，模板结构如以下代码所示。

> **说明** 关于Terraform类型模板组成部分的更多信息，请参见模板语法。

```
ROSTemplateFormatVersion: '2015-09-01'
Transform: 'Aliyun::Terraform-v1.0'
Parameters:
  subnet_mask:
    Type: Number
    Description:
      en: Subnet mask of VSwitch
      zh-cn: 交换机子网掩码
    Label:
      en: Subnet mask
      zh-cn: 子网掩码
```

```
    MinValue: 13
    MaxValue: 31
    Default: 21
Outputs:
  vpc_id:
    Description:
      en: VPC ID
      zh-cn: 专有网络ID
Workspace:
  main.tf: |-
    variable "zone_id" {
      type = string
      description = <<EOT
      {
        "AssociationProperty": "ALIYUN::ECS::Instance::ZoneId",
        "Description": {
          "en": "Zone of VSwitch",
          "zh-cn": "交换机所在可用区"
        },
        "Label": {
          "en": "Zone",
          "zh-cn": "可用区"
        }
      }
      EOT
    }
    variable "subnet_mask" {
      type = number
    }
    module "my_vpc" {
      source      = "./modules/vpc"
    }
    resource "alicloud_vswitch" "vsw" {
      vpc_id            = "${module.my_vpc.vpc_id}"
      cidr_block        = "172.16.0.0/${var.subnet_mask}"
      availability_zone = var.zone_id
    }
    output "vsw_id" {
      value = "${alicloud_vswitch.vsw.id}"
      description = <<EOT
      {
        "Description": {
          "en": "VSwitch ID",
          "zh-cn": "交换机ID"
        }
      }
      EOT
    }
  modules/vpc/main.tf: |-
    variable "vpc_name" {
      type = string
      default = "tf_test"
      description = "专有网络名称"
    }
```

```
resource "alicloud_vpc" "vpc" {
  name       = var.vpc_name
  cidr_block = "172.16.0.0/12"
}
output "vpc_id" {
  value = "${alicloud_vpc.vpc.id}"
}
```

ROSTemplateFormatVersion（必选）

ROS支持的模板版本号。取值：2015-09-01。

Transform（必选）

ROS支持的Terraform版本。取值：

- Aliyun::Terraform-v0.12：Terraform 0.12版本。
- Aliyun::Terraform-v0.15：Terraform 0.15版本。
- Aliyun::Terraform-v1.0：Terraform 1.0版本。
- Aliyun::Terraform-v1.1：Terraform 1.1版本。

Transform使用说明如下：

- ROS会随Terraform版本发布增加Transform参数的取值。
- Terraform小版本变动（版本号x.y.z中的z发生变化）不影响Transform的取值。
- 满足如下条件时，Transform参数可以通过资源栈继续创建功能或更新功能修改，否则不能修改。

初始值	修改后的值
Aliyun::Terraform-v1.0	Aliyun::Terraform-v1.1
Aliyun::Terraform-v1.1	Aliyun::Terraform-v1.0

> **说明** 您可以调用GetFeatureDetails接口，获取每个Transform版本允许更新到的版本（UpdateAllowedTransforms参数）。

Workspace（必选）

Terraform Workspace中所有模块的键值对。键为模块文件路径，值为模块文件内容。

Workspace使用说明如下：

- Workspace内容不能为空，且最多指定50个文件。
- 文件路径
  - 最长为1024个字符，路径中每个文件夹或文件的名字最长为255个字符。
  - 文件路径必须是相对路径，不能以正斜线（/）开头，不能以 `.json` 、 `.tfstate` 或 `.hcl` 结尾。如果.tf文件以.debug.tf结尾，则该文件会被ROS忽略，不参与Terraform编排。
  - 可包含英文字母、数字或特殊字符 `!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~` 。
  - 最大深度为5。例如：main.tf深度为1，modules/vpc/main.tf深度为3。
  - 路径分隔符正斜线（/）之间的值不能为空、 `.` 或 `..` 。
- 文件内容

- 不能使用Provisioner功能、Backend功能和Terraform Cloud功能。
- 可以使用Module Source功能，但只能为Workspace内的相对引用。必须以 `./` 开头，路径分隔符正斜线（/）之间的值不能为空、 `.` 或 `..` 。
- 可以使用Provider功能。
  以下代码示例中可变参数均需替换为您实际的相关信息，尖括号（<>）在最终代码中不保留。涉及替换的可变参数取值如下：
  - <provider>：alicloud、aws、azurerm、random、template、time、fortios、fortimanager、helm或kubernetes。
  - <host>：registry.terraform.io或未定义。
  - <namespace>：hashicorp、aliyun、fortinetdev或未定义。

```
terraform {
    required_providers {
        <provider> = {
            source = "<host>/<namespace>/<provider>"
            ...
        }
    }
}
provider "<provider>" {
    ...
}
```

- 可以使用Provider中包含的 `Resources` 和 `Data Sources` ，不能使用terraform_remote_state（ `Data Sources` 的一种）、template.template_dir（ `Resources` 的一种）。
- 函数file、fileexists、fileset、filebase64和templatefile的path参数限制如下：
  - 必须存在。
  - 必须是不跨行的字符串，不能是其他形式，例如：变量的引用。
  - 使用正斜线（/）分隔后，第一个分词必须是 `${path.module}` 、 `${path.root}` 、 `${path.cwd}` 或 `${terraform.workspace}` 。
  - 使用正斜线（/）分隔后，从第二个分词开始，每个分词中只能包含英文字母、数字或特殊字符 `-_.` ，不能是空、 `.` 或 `..` 。

Description（可选）

Terraform类型模板的描述信息。

Parameters（可选）

Terraform类型模板的参数，与ROS类型模板参数（Parameters）语法相同。更多信息，请参见概览。

Parameters使用说明如下：

- Parameters中的参数必须在.tf文件中定义，但.tf文件中的参数可以不在Parameters中定义。
  - 如果.tf文件中定义的参数未在Parameters中定义，则ROS会自动从.tf文件中提取参数，并在Parameters中生成定义。
  - 如果.tf文件中定义的参数已经在Parameters中定义，则不会提取参数，使用Parameters中已有的定义。
- Parameters中的参数类型必须与.tf文件中的参数类型一致。
  假定Parameters中参数类型为A，.tf文件中的参数类型为B，其约束关系如下表所示。

Parameters与.tf文件中的参数类型的约束关系

.tf文件中的参数类型（B）	Parameters中的参数类型（A）
any或者未定义	假定.tf文件中的参数的默认值的类型为C，A的取值如下： ○ 如果C未定义或默认值为null：`String`、`Number`、`CommaDelimitedList`、`Json`、`Boolean`、`ALIYUN::OOS::Parameter::Value`或`ALIYUN::OOS::SecretParameter::Value`。 ○ 如果C为string：`String`、`ALIYUN::OOS::Parameter::Value`或`ALIYUN::OOS::SecretParameter::Value`。 ○ 如果C为number：`Number`、`String`、`ALIYUN::OOS::Parameter::Value`或`ALIYUN::OOS::SecretParameter::Value`。 ○ 如果C为bool：`Boolean`、`String`、`ALIYUN::OOS::Parameter::Value`或`ALIYUN::OOS::SecretParameter::Value`。 ○ 如果C为list（string）：`Json`、`CommaDelimitedList`、`String`、`ALIYUN::OOS::Parameter::Value`或`ALIYUN::OOS::SecretParameter::Value`。 ○ 如果C为其他类型：`Json`、`String`、`ALIYUN::OOS::Parameter::Value`或`ALIYUN::OOS::SecretParameter::Value`。
string	`String`、`ALIYUN::OOS::Parameter::Value`或`ALIYUN::OOS::SecretParameter::Value`。
number	`Number`、`String`、`ALIYUN::OOS::Parameter::Value`或`ALIYUN::OOS::SecretParameter::Value`。
bool	`Boolean`、`String`、`ALIYUN::OOS::Parameter::Value`或`ALIYUN::OOS::SecretParameter::Value`。
list（string）	`Json`、`CommaDelimitedList`、`String`、`ALIYUN::OOS::Parameter::Value`或`ALIYUN::OOS::SecretParameter::Value`。
其他类型	`Json`、`String`、`ALIYUN::OOS::Parameter::Value`或`ALIYUN::OOS::SecretParameter::Value`。

您可以根据需要，提取Terraform模板参数。如果.tf文件中定义的参数未在Parameters中定义，则ROS会自动从.tf文件中提取参数，并在Parameters中生成定义。

- 参数名：提取前后参数名不变。除了内置的伪参数外，参数名不能以**ALIYUN__**开头。
- 参数传递文件：不建议再使用.tfvars文件（Terraform用来传递参数值的文件）传递参数值，而是通过ROS参数进行传递。如果要使用.tfvars文件，只能使用.auto.tfvars文件或terraform.tfvars文件（其他.tfvars文件无效），且需要注意.tfvars文件与ROS参数之间的优先级。优先级从高到低如下：
  i. .auto.tfvars文件。
  不建议使用多个文件。如果有多个文件，按文件名逆序排列，排前面的优先级高。例如：b.auto.tfvars比a.auto.tfvars优先级高。
  ii. ROS参数。
  iii. terraform.tfvars文件。
- 伪参数：通过在.tf文件中定义如下参数，您可以使用ROS相应的伪参数（Pseudo parameters）。

.tf文件中参数名	.tf文件中参数类型	ROS伪参数名
ALIYUN__StackId	string	ALIYUN::StackId
ALIYUN__StackName	string	ALIYUN::StackName
ALIYUN__TenantId	string	ALIYUN::TenantId
ALIYUN__Region	string	ALIYUN::Region
ALIYUN__AccountId	string	ALIYUN::AccountId
ALIYUN__NoValue	string	ALIYUN::NoValue

- 提取规则：

.tf文件中参数原有字段	提取后的Parameters中参数的字段及说明

.tf文件中参数原有字段	提取后的Parameters中参数的字段及说明
type	Type。规则如下： ○ 如果type为any或者未定义，则基于默认值推断： ■ 如果没有定义默认值或者默认值为null，则Type为 `String` 。 ■ 如果默认值为string类型，则Type为 `String` 。 ■ 如果默认值为number类型，则Type为 `Number` 。 ■ 如果默认值为bool类型，则Type为 `Boolean` 。 ■ 如果默认值为其他类型，则Type为 `Json` 。 **注意** ■ 强烈建议对类型进行定义，避免默认类型（String）与预期不一致。 ■ 如果type未定义且默认值形式为数值时，Terraform会把它识别为字符串，ROS推断出的类型（Type）为String。 `variable "i" { default = 1 //会被识别为字符串："1"，ROS推断类型为String。 } variable "f" { default = 1.1 //会被识别为字符串："1.1"，ROS推断类型为String。 } variable "l" { default = [1.1] //会被识别为字符串：["1.1"]，ROS推断类型为Json。 }` ○ 如果type为string，则Type为 `String` 。 ○ 如果type为number，则Type为 `Number` 。 ○ 如果type为bool，则Type为 `Boolean` 。 ○ 如果type为其他类型，则Type为 `Json` 。

.tf文件中参数原有字段	提取后的Parameters中参数的字段及说明
default	Default
sensitive	NoEcho
description	○ 如果description不是 `Json` 字符串，则生成Description。 ○ 如果description是 `Json` 字符串，则要求其内容与ROS类型模板参数（Parameters）语法相同，限制如下： ■ 不能包含不支持的字段。 ■ 如果NoEcho已定义，则不再使用sensitive；如果NoEcho未定义，则使用sensitive。 ■ 如果Default已定义，则不再使用default；如果Default未定义，则使用default。 ■ 如果Type已定义，则必须与type一致，规则参见Parameters与.tf文件中的参数类型的约束关系。

- 提取后的参数示例：

```
Parameters:
  subnet_mask:
    Description:
      en: Subnet mask of VSwitch
      zh-cn: 交换机子网掩码
    Label:
      en: Subnet mask
      zh-cn: 子网掩码
    MaxValue: 31
    MinValue: 13
    Default: 21
    Type: Number
  zone_id:
    AssociationProperty: 'ALIYUN::ECS::Instance::ZoneId'
    Description:
      en: Zone of VSwitch
      zh-cn: 交换机所在可用区
    Label:
      en: Zone
      zh-cn: 可用区
    Type: String
  vpc_name:
    Default: tf_test
    Description: 专有网络名称
    Type: String
```

Outputs（可选）

Terraform类型模板的输出，与ROS类型模板输出（Outputs）语法相同。

Outputs使用说明如下：

- Outputs中的输出必须在.tf文件中定义，但.tf文件中的输出可以不在Outputs中定义。

  - 如果.tf文件中定义的输出未在Outputs中定义，则ROS会自动从.tf文件中提取输出，并在Outputs中生成定义。
  - 如果.tf文件中定义的输出已经在Outputs中定义，则不会提取输出，而是使用Outputs中已有的定义。
- Outputs中无法使用Condition字段。
- Value字段建议设置为null，实际会返回Terraform的输出。

您可以根据需要，提取Terraform模板输出。如果.tf文件中定义的输出未在Outputs中定义，则ROS会自动从.tf文件中提取输出，并在Outputs中生成定义。

- 提取规则：
  - 输出名称：保持不变。
  - description字段：
    - 如果description不是 `Json` 字符串，则对应生成Description字段。
    - 如果description是 `Json` 字符串，则要求其内容与ROS类型模板输出（Outputs）语法相同，目前只支持Description字段。
- 提取后的输出示例：

```
Outputs:
  vpc_id:
    Value: null
    Description:
      en: VPC ID
      zh-cn: 专有网络ID
  vsw_id:
    Value: null
    Description:
      en: VSwitch ID
      zh-cn: 交换机ID
```

Metadata（可选）

关于模板元数据的更多信息，请参见元数据（Metadata）。

> **说明** 如果仅在控制台使用Metadata，您可以通过在Workspace中添加 `.metadata` 文件代替。

Mappings（可选）

关于模板映射的更多信息，请参见映射（Mappings）。

> **说明** 如果仅在控制台使用Mappings，您可以通过在Workspace中添加 `.mappings` 文件代替。

9.5. Terraform支持的功能和资源

资源编排服务ROS（Resource Orchestration Service）为Terraform提供了托管的能力，本文为您介绍Terraform对ROS功能和资源的支持情况。

ROS版本支持情况

ROS当前支持的Terraform版本和Provider版本如下表所示。

Terraform版本	Provider版本
0.12.28	• alicloud：1.121.2 • aws：3.37.0 • azurerm：2.56.0 • random：3.1.0 • template：2.2.0 • time：0.7.0
0.15.3	• alicloud：1.123.0 • aws：3.42.0 • azurerm：2.59.0 • random：3.1.0 • template：2.2.0 • time：0.7.1
1.0.11	• alicloud：1.139.0~1.165.0 • aws：3.63.0~4.13.0 • azurerm：2.81.0~3.5.0 • random：3.1.0~3.1.3 • template：2.2.0 • time：0.7.2 • fortios：1.13.2~1.14.1 • fortimanager：1.3.4~1.4.0 • helm：2.3.0~2.5.1 • kubernetes：2.6.1~2.11.0
1.1.9	• alicloud：1.139.0~1.165.0 • aws：3.63.0~4.13.0 • azurerm：2.81.0~3.5.0 • random：3.1.0~3.1.3 • template：2.2.0 • time：0.7.2 • fortios：1.13.2~1.14.1 • fortimanager：1.3.4~1.4.0 • helm：2.3.0~2.5.1 • kubernetes：2.6.1~2.11.0

说明 ROS会持续更新支持的Terraform版本和Provider版本。您可以调用GetFeatureDetails接口，获取支持的Terraform版本列表。

ROS功能支持情况

功能	支持的功能	不支持的功能
资源栈	• 预览、创建、更新、删除和查询资源栈。 • 查询资源、事件（包括资源栈和资源）、输出和模板。 • 通过资源栈创建资源时，为资源栈中的部分资源添加系统标签acs:ros:stackId。关于资源的更多信息，请参见ROS资源支持情况章节的**支持系统标签的资源**列表。 • 通过资源栈创建或更新资源时，向资源栈中的部分资源传递资源栈的用户标签。关于资源的更多信息，请参见ROS资源支持情况章节的**支持传递资源栈用户标签的资源**列表。 • 通过资源栈创建或更新资源时，向资源栈中的部分资源传递资源栈所属的资源组。关于资源的更多信息，请参见ROS资源支持情况章节的**支持传递资源栈所属资源组的资源**列表。 • 设置超时时间（10~120分钟）、设置参数和输出、失败继续创建、状态通知、删除保护、删除资源栈时保留所有资源、删除资源栈时保留部分资源、设置RAM角色、管理标签和管理资源组。 • 管理更改集。 • 偏差检测。	失败回滚、资源栈策略、替换更新、取消更新、偏差纠正、信号通知、资源导入和风险检查等。
资源栈组	• 创建、更新、删除、查询资源栈组。 • 创建、更新、删除、查询资源栈实例，查询、停止资源栈操作。 • 管理标签和管理资源组。 • 偏差检测。	无。
模板	• 创建、更新、删除、查询、共享和校验模板。 • 对模板中涉及的收费资源进行询价。关于资源的更多信息，请参见ROS资源支持情况章节的**支持询价的资源**列表。 • 管理标签和管理资源组。	生成模板的RAM策略、查询模板参数取值。
其他	• STS（Security Token Service）。 • 查询服务开通状态和服务角色信息。 • 查询功能详情。	查询资源类型。

ROS API支持情况

功能	支持的API
资源栈	PreviewStack、CreateStack、ContinueCreateStack、UpdateStack、DeleteStack、GetStack、ListStacks、ListStackResources、GetStackResource、ListStackEvents和SetDeletionProtection。 **说明** GetStack和ListStacks中StackType取值为Terraform时，说明资源栈类型是Terraform。
更改集	CreateChangeSet、ExecuteChangeSet、DeleteChangeSet、GetChangeSet和ListChangeSets。
偏差检测	DetectStackDrift、DetectStackGroupDrift、GetStackDriftDetectionStatus和ListStackResourceDrifts。 **说明** 不支持使用 DetectStackResourceDrift 对单个资源进行同步的偏差检测。
资源栈组	CreateStackGroup、UpdateStackGroup、DeleteStackGroup、GetStackGroup、ListStackGroups、CreateStackInstances、UpdateStackInstances、DeleteStackInstances、GetStackInstance、ListStackInstances、StopStackGroupOperation、GetStackGroupOperation、ListStackGroupOperations和ListStackGroupOperationResults。
模板	CreateTemplate、UpdateTemplate、DeleteTemplate、GetTemplate、ListTemplates、ListTemplateVersions、SetTemplatePermission、ValidateTemplate和GetTemplateEstimateCost。
标签	TagResources、UntagResources、ListTagKeys、ListTagValues和ListTagResources。
资源组	MoveResourceGroup。
其他	GetServiceProvisions、GetFeatureDetails。

ROS资源支持情况

在ROS中，Terraform支持的主流云厂商资源如下：

- 阿里云资源

> 说明
> - 支持Terraform在线调试工具。
> - ROS提供了一个默认的Provider，使用当前账号的临时AccessKey（或STS凭证）以及资源栈所属的地域ID。

其中，部分资源支持询价、系统标签、传递资源栈用户标签和传递资源栈所属资源组，具体如下：

> **说明** 您可以调用GetFeatureDetails接口，获取支持询价、系统标签、传递资源栈用户标签和传递资源栈所属资源组的资源类型列表。

- 支持询价的资源
  - ECS：alicloud_instance、alicloud_ecs_disk、alicloud_disk和alicloud_ecs_dedicated_host。
  - VPC：alicloud_eip_address、alicloud_eip、alicloud_common_bandwidth_package、alicloud_nat_gateway和alicloud_vpn_gateway。
  - SLB：alicloud_slb_load_balancer和alicloud_slb。
  - RDS：alicloud_db_instance和alicloud_db_readonly_instance。
  - Redis：alicloud_kvstore_instance。
  - PolarDB：alicloud_polardb_cluster。
  - MongoDB：alicloud_mongodb_instance。
  - CEN：alicloud_cen_bandwidth_package。
  - MarketPlace：alicloud_market_order。
  - PolarDB-X：alicloud_drds_instance。
  - ECI：alicloud_eci_container_group。
  - EMR：alicloud_emr_cluster。
  - Elasticsearch：alicloud_elasticsearch_instance。
  - SAE：alicloud_sae_application。
  - GPDB：alicloud_gpdb_elastic_instance和alicloud_gpdb_instance。
- 支持系统标签的资源
  - ECS：alicloud_instance、alicloud_ecs_disk、alicloud_disk、alicloud_ecs_dedicated_host、alicloud_security_group和alicloud_key_pair、alicloud_ecs_launch_template、alicloud_ecs_network_interface、alicloud_image_copy、alicloud_image、alicloud_ecs_snapshot和alicloud_snapshot。
  - VPC：alicloud_eip_address、alicloud_eip、alicloud_common_bandwidth_package、alicloud_nat_gateway、alicloud_vpn_gateway、alicloud_vpc和alicloud_vswitch。
  - SLB：alicloud_slb_load_balancer和alicloud_slb。
  - RDS：alicloud_db_instance和alicloud_db_readonly_instance。
  - Redis：alicloud_kvstore_instance。
  - PolarDB：alicloud_polardb_cluster。
  - MongoDB：alicloud_mongodb_instance。
  - Elasticsearch：alicloud_elasticsearch_instance。
  - GPDB：alicloud_gpdb_elastic_instance和alicloud_gpdb_instance。

- 支持传递资源栈用户标签的资源
  - ECS：alicloud_instance、alicloud_ecs_disk、alicloud_disk、alicloud_ecs_dedicated_host、alicloud_security_group、alicloud_key_pair、alicloud_ecs_launch_template、alicloud_ecs_network_interface、alicloud_image_copy、alicloud_image、alicloud_ecs_snapshot和alicloud_snapshot。
  - VPC：alicloud_eip_address、alicloud_eip、alicloud_common_bandwidth_package、alicloud_nat_gateway、alicloud_vpn_gateway、alicloud_vpc和alicloud_vswitch。
  - SLB：alicloud_slb_load_balancer和alicloud_slb。
  - RDS：alicloud_db_instance和alicloud_db_readonly_instance。
  - Redis：alicloud_kvstore_instance。
  - PolarDB：alicloud_polardb_cluster。
  - MongoDB：alicloud_mongodb_instance。
  - CEN：alicloud_cen_bandwidth_package。
  - PolarDB-X：alicloud_drds_instance。
  - EMR：alicloud_emr_cluster。
  - Elasticsearch：alicloud_elasticsearch_instance。
  - GPDB：alicloud_gpdb_elastic_instance和alicloud_gpdb_instance。
- 支持传递资源栈所属资源组的资源
  - ECS：alicloud_instance、alicloud_ecs_disk、alicloud_disk、alicloud_ecs_dedicated_host、alicloud_security_group、alicloud_key_pair、alicloud_ecs_launch_template、alicloud_ecs_network_interface、alicloud_image_copy、alicloud_image、alicloud_snapshot和alicloud_ecs_snapshot。
  - VPC：alicloud_vpc、alicloud_common_bandwidth_package、alicloud_eip_address和alicloud_eip。
  - SLB：alicloud_slb_load_balancer和alicloud_slb。
  - RDS：alicloud_db_readonly_instance。
  - Redis：alicloud_kvstore_instance。
  - PolarDB：alicloud_db_instance和alicloud_polardb_cluster。
  - MongoDB：alicloud_mongodb_instance。
  - ECI：alicloud_eci_container_group。
  - PolarDB-X：alicloud_drds_instance。
  - EMR：alicloud_emr_cluster。
  - Elasticsearch：alicloud_elasticsearch_instance。

- AWS资源
- Azure资源

9.6. Terraform代码开发方式和建议

Terraform是一个开源的自动化资源编排工具。资源编排服务ROS（Resource Orchestration Service）为Terraform提供了托管的能力。当您了解了Terraform和Terraform托管方式，需要开发Terraform代码并在ROS中使用时，可以采用本文介绍的开发方式和开发建议。

开发方式

建议使用熟悉的开发方式编写和测试Terraform代码。您可以采用以下开发方式：

- 本地开发。
- 使用Terraform在线调试工具开发（仅支持阿里云）。
- 使用ROS创建Terraform类型资源栈，然后根据需求继续创建或更新资源栈。更多信息，请参见创建Terraform类型资源栈、继续创建资源栈和更新资源栈。

开发建议

- **不建议在.tf文件中声明阿里云（alicloud）Provider。**
  ROS提供了一个默认的阿里云Provider，使用当前阿里云账号的临时AccessKey（或STS凭证）以及资源栈所属的地域。使用默认Provider的优点如下：
  - 简化开发，提升安全性，降低访问密钥（AccessKey）泄露的风险。
  - 保证资源与资源栈所属地域相同，以便进行统一管理和集成。
  - 当资源与资源栈所属地域相同时，支持询价、系统标签、传递资源栈用户标签、传递资源栈所属资源组等功能。
- **把仅供本地使用的代码放入以.debug.tf结尾的文件。**
  在Terraform托管中，ROS会忽略以.debug.tf结尾的文件，不参与Terraform编排。但在本地测试时，会参与Terraform编排。例如：您可以编写一个名为provider.debug.tf的文件，对阿里云Provider进行配置。在本地开发时，该文件中配置会生效，资源会创建在中国香港（cn-hongkong）地域。但在ROS中创建资源栈时，ROS会忽略该文件，资源会创建在资源栈所属地域。provider.debug.tf文件内容如下：

  ```
  variable "region" {
    type = string
    default = "cn-hongkong"
  }
  provider "alicloud" {
    region ="${var.region}"
  }
  ```

- **建议指定Provider版本。**
  从1.0版本（Aliyun::Terraform-v1.0）开始，Terraform托管支持连续的Provider版本。通过指定Provider版本，可以防止Provider更新引入问题，保障稳定性。代码示例如下：

  ```
  terraform {
    required_providers {
      alicloud = {
        source  = "aliyun/alicloud"
        version = "1.140.0"
      }
    }
  }
  ```

  关于Provider版本的更多信息，请参见ROS版本支持情况的**Provider版本**列。
- **不建议使用.tfvars文件，而是通过ROS参数传递变量值。**
  其优点如下：
  - 减少模板的修改次数。大部分情况下，只需要修改参数值。
  - 变量与ROS参数一一对应，在控制台清晰可见。如果使用.tfvars文件，可能覆盖变量值，造成实际值与控制台显示值不一致。

  更多信息，请参见Parameters（可选）。
- **通过伪参数获取资源栈信息。**

更多信息，请参见Parameters（可选）。例如：在.tf文件中定义变量ALIYUN__Region，通过var.ALIYUN__Region访问，即可获取资源栈所属地域。代码示例如下：

```
variable "ALIYUN__Region" {
  type = string
  default = "cn-hongkong"
}
```

- **细化变量定义。**
  ROS会自动把Terraform变量转换成ROS参数，细化变量定义会使得ROS转换的结果更为准确。更多信息，请参见Parameters（可选）。
  - 为变量设置type字段，否则ROS可能会将变量当作字符串类型处理，并传递给Terraform，Terraform在编排时可能出现变量类型错误。
  - 如果参数包含敏感信息，在其对应的变量中将sensitive设置为true。

    ```
    variable "password" {
      type = string
      sensitive = true
    }
    ```

- **使用Metadata控制参数（变量）在控制台的显示。**
  - 为参数分组：更多信息，请参见元数据（Metadata）和使用Metadata为参数分组。
  - 隐藏参数：使用Metadata.ALIYUN::ROS::Interface.Hidden指定需要隐藏的参数列表。

    ```
    {
      "ROSTemplateFormatVersion": "2015-09-01",
      "Description": "Creates a simple oss bucket",
      "Parameters": {
        "BucketName": {
          "Type": "String",
          "Label": "Bucket Name",
          "Description": {
            "en": "Bucket name",
            "zh-cn": "Bucket名称"
          },
          "Default": "bucketName1"
        }
      },
      "Metadata": {
        "ALIYUN::ROS::Interface": {
          "Hidden": [
            "BucketName"
          ]
        }
      },
      "Workspace": ...
    }
    ```

- **控制参数（变量）在控制台的输入方式。**
  - 在ROS参数中，通过AssociationProperty和AssociationPropertyMetadata可以自动验证参数值的合法性，并且给参数提供取值信息。更多信息，请参见AssociationProperty和AssociationPropertyMetadata和在资源编排控制台动态选择参数配置。

- 在Terraform变量中，通过description字段控制AssociationProperty和AssociationPropertyMetadata。更多信息，请参见Parameters（可选）。代码示例如下：

```
variable "vpc_id" {
  type = string
  description = <<EOT
  {
    "AssociationProperty": "ALIYUN::ECS::VPC::VPCId",
    "Description": {
      "en": "Please search the ID starts with (vpc-xxx)from console-Virtual Private Clo
ud",
      "zh-cn": "在专有网络控制台的专有网络页面查看专有网络ID。"
    },
    "Label": {
      "en": "Existing VPC ID",
      "zh-cn": "现有专有网络ID"
    }
  }
  EOT
}
```

9.7. Terraform模板示例

本文为您提供常见的Terraform模板示例。

创建VPC类型的ECS实例

以下模板将创建专有网络VPC、交换机vSwitch、安全组SecurityGroup，以及VPC类型的ECS实例。

```
variable "name" {
  default = "auto_provisioning_group"
}
variable "vpc_cidr_block" {
  default = "172.16.0.0/16"
}
variable "vsw_cidr_block" {
  default = "172.16.0.0/24"
}
# Create a new ECS instance for a VPC
resource "alicloud_security_group" "group" {
  name        = "tf_test_foo"
  description = "foo"
  vpc_id      = alicloud_vpc.vpc.id
}
resource "alicloud_kms_key" "key" {
  description             = "Hello KMS"
  pending_window_in_days = "7"
  key_state               = "Enabled"
}
data "alicloud_zones" "default" {
  available_disk_category     = "cloud_efficiency"
  available_resource_creation = "VSwitch"
}
# Create a new ECS instance for VPC
resource "alicloud_vpc" "vpc" {
```

```
  name       = var.name
  cidr_block = var.vpc_cidr_block
}
resource "alicloud_vswitch" "vswitch" {
  vpc_id            = alicloud_vpc.vpc.id
  cidr_block        = var.vsw_cidr_block
  zone_id           = data.alicloud_zones.default.zones[0].id
  vswitch_name      = var.name
}
resource "alicloud_instance" "instance" {
  # cn-beijing
  availability_zone = data.alicloud_zones.default.zones[0].id
  security_groups   = alicloud_security_group.group.*.id
  # series III
  instance_type              = "ecs.n4.large"
  system_disk_category       = "cloud_efficiency"
  system_disk_name           = "test_foo_system_disk_name"
  system_disk_description    = "test_foo_system_disk_description"
  image_id                   = "ubuntu_18_04_64_20G_alibase_20190624.vhd"
  instance_name              = "test_foo"
  vswitch_id                 = alicloud_vswitch.vswitch.id
  internet_max_bandwidth_out = 10
  data_disks {
    name        = "disk2"
    size        = 20
    category    = "cloud_efficiency"
    description = "disk2"
    encrypted   = true
    kms_key_id  = alicloud_kms_key.key.id
  }
}
```

创建VPC类型的RDS实例

以下模板将创建专有网络VPC、交换机vSwitch，以及VPC类型的RDS实例。

```
variable "name" {
  default = "tf-testaccdbinstance"
}
variable "creation" {
  default = "Rds"
}
data "alicloud_zones" "example" {
  available_resource_creation = var.creation
}
resource "alicloud_vpc" "example" {
  name       = var.name
  cidr_block = "172.16.0.0/16"
}
resource "alicloud_vswitch" "example" {
  vpc_id     = alicloud_vpc.example.id
  cidr_block = "172.16.0.0/24"
  zone_id    = data.alicloud_zones.example.zones[0].id
  name       = var.name
}
resource "alicloud_db_instance" "example" {
  engine               = "MySQL"
  engine_version       = "5.6"
  instance_type        = "rds.mysql.s2.large"
  instance_storage     = "30"
  instance_charge_type = "Postpaid"
  instance_name        = var.name
  vswitch_id           = alicloud_vswitch.example.id
  monitoring_period    = "60"
}
```

创建SLB实例

以下模板将创建专有网络VPC、交换机vSwitch，以及VPC类型的SLB实例。

```
variable "name" {
  default = "terraformtestslbconfig"
}
data "alicloud_zones" "default" {
  available_resource_creation = "VSwitch"
}
resource "alicloud_vpc" "default" {
  name       = var.name
  cidr_block = "172.16.0.0/12"
}
resource "alicloud_vswitch" "default" {
  vpc_id            = alicloud_vpc.default.id
  cidr_block        = "172.16.0.0/21"
  zone_id           = data.alicloud_zones.default.zones[0].id
  vswitch_name      = var.name
}
resource "alicloud_slb" "default" {
  name          = var.name
  specification = "slb.s2.small"
  vswitch_id    = alicloud_vswitch.default.id
}
```

更多示例

关于Terraform模板示例的更多信息，请参见Terraform User Guide。

10.资源导入

10.1. 概览

您可以将现有资源通过资源导入的方式，导入到ROS的资源栈中，统一进行资源管理和编排。

支持资源导入的资源类型

关于资源导入支持的资源类型，请参见支持偏差检测和资源导入的资源类型。

资源导入方式

现有资源导入到ROS，有以下两种方式：

场景	描述
使用现有资源创建资源栈	如果您想将资源导入到一个新的资源栈中，可以通过该方式，导入资源的同时会新创建一个资源栈。
将现有资源导入资源栈	如果您想将资源导入到一个已创建的资源栈中，可以通过该方式，选择目标资源栈直接导入资源。

资源导入使用说明

资源导入时，您需要关注以下内容：

- 请确保模板内容正确。
  - 模板中包含资源栈原有资源和待导入资源，待导入资源为已存在的资源。待导入资源必须设置 `DeletionPolicy` 属性，为防止资源被误删除，请将 `DeletionPolicy` 设置为 `Retain` 。
  - 资源类型支持待导入资源的属性和取值。
  - 模板中指定了每种资源类型必须的属性。
- 需要获取资源标识符属性和资源标识符值。关于如何获取资源标识符属性，请参见获取待导入资源的标识符属性。
- 资源导入完成后，建议您对导入的资源执行偏差检测。偏差检测可以确保模板配置与实际配置匹配。具体操作，请参见检测资源栈的偏差状态。
- 资源导入操作遵循资源栈相关限制。更多信息，请参见使用限制。
- 针对以下资源，不支持将同一资源导入到多个资源栈中。

云服务	资源类型
API网关（API Gateway）	◦ ALIYUN::ApiGateway::Api ◦ ALIYUN::ApiGateway::App ◦ ALIYUN::ApiGateway::Group
云解析DNS（Alibaba Cloud DNS）	ALIYUN::DNS::DomainRecord
云服务器ECS（Elastic Compute Service）	◦ ALIYUN::ECS::Snapshot ◦ ALIYUN::ECS::VPC ◦ ALIYUN::ECS::VSwitch

云服务	资源类型
弹性伸缩（Auto Scaling）	○ ALIYUN::ESS::ScalingConfiguration ○ ALIYUN::ESS::ScalingRule
函数计算FC（Function Compute）	○ ALIYUN::FC::CustomDomain ○ ALIYUN::FC::Function ○ ALIYUN::FC::Service ○ ALIYUN::FC::Trigger
云数据库MongoDB版（ApsaraDB for MongoDB）	ALIYUN::MONGODB::Instance
文件存储NAS（Network Attached Storage ）	ALIYUN::NAS::FileSystem
阿里云关系型数据库RDS（Relational Database Service）	ALIYUN::RDS::DBInstance
云数据库Redis版（ApsaraDB for Redis）	ALIYUN::REDIS::Instance
负载均衡SLB（Server Load Balancer）	ALIYUN::SLB::MasterSlaveServerGroup
日志服务SLS（Log Service）	ALIYUN::SLS::Project
专有网络VPC（Virtual Private Cloud）	○ ALIYUN::VPC::NatGateway ○ ALIYUN::VPC::SnatEntry

状态码

资源导入时的状态码如下：

- 资源栈的状态码

状态码	描述
IMPORT_CREATE_IN_PROGRESS	正在通过资源导入创建资源栈。
IMPORT_CREATE_FAILED	通过资源导入创建资源栈失败。
IMPORT_CREATE_COMPLETE	通过资源导入创建资源栈成功。
IMPORT_CREATE_ROLLBACK_IN_PROGRESS	通过资源导入创建资源栈失败，正在回滚。
IMPORT_CREATE_ROLLBACK_FAILED	通过资源导入创建资源栈失败，回滚失败。

状态码	描述
IMPORT_CREATE_ROLLBACK_COMPLETE	通过资源导入创建资源栈失败，回滚成功。
IMPORT_UPDATE_IN_PROGRESS	正在通过资源导入更新资源栈。
IMPORT_UPDATE_FAILED	通过资源导入更新资源栈失败。
IMPORT_UPDATE_COMPLETE	通过资源导入更新资源栈成功。
IMPORT_UPDATE_ROLLBACK_IN_PROGRESS	通过资源导入更新资源栈失败，正在回滚。
IMPORT_UPDATE_ROLLBACK_FAILED	通过资源导入更新资源栈失败，回滚失败。
IMPORT_UPDATE_ROLLBACK_COMPLETE	通过资源导入更新资源栈失败，回滚成功。

- 资源的状态码

状态码	描述
IMPORT_IN_PROGRESS	资源正在导入中。
IMPORT_FAILED	资源导入失败。
IMPORT_COMPLETE	资源导入成功。

10.2. 获取待导入资源的标识符属性

资源导入时，需要提供资源标识符属性。本文以ALIYUN::VPC::EIP为例，为您介绍如何调用GetResourceTypeTemplate和GetTemplateSummary接口，获取待导入资源的标识符属性。

操作步骤

1. 登录OpenAPI开发者门户。
2. 调用GetResourceTypeTemplate接口查询ALIYUN::VPC::EIP的模板（TemplateBody）。
   i. 搜索GetResourceTypeTemplate接口。
   ii. 将**ResourceType**设置为ALIYUN::VPC::EIP，然后单击**发起调用**。

   > **说明** 更多资源类型，请参见资源类型索引。

   调用结果如下：

```
{
 "RequestId": "4EE61317-00F7-4DB6-9FBD-E12ECC79805A",
 "TemplateBody": {
  "Parameters": {
   "Description": {
    "Type": "String",
```

```
    "Description": "Optional. The description of the EIP. The description must be 2
to 256 characters in length. It must start with a letter. It cannot start with http
://  or https://."
   },
   "ResourceGroupId": {
    "Type": "String",
    "Description": "Resource group id."
   },
   "InstanceChargeType": {
    "Type": "String",
    "Description": "The resource charge type. Default value is Postpaid",
    "AllowedValues": [
     "Prepaid",
     "Postpaid"
    ],
    "Default": "Postpaid"
   },
   "PricingCycle": {
    "Type": "String",
    "Description": "Price cycle of the resource. This property has no default value
. If ChargeType is specified as Postpaid, this value will be ignore.",
    "AllowedValues": [
     "Month",
     "Year"
    ],
    "Default": "Month"
   },
   "Isp": {
    "Type": "String",
    "Description": "ISP tag for finance cloud region. only for cn-hangzhou and cn-q
ingdao region), if you are not finance cloud user, this value will be ignore."
   },
   "Period": {
    "Type": "Number",
    "Description": "Prepaid time period. While choose by pay by month, it could be
from 1 to 9. While choose pay by year, it could be from 1 to 3.",
    "MinValue": 1,
    "MaxValue": 9,
    "Default": 1
   },
   "DeletionProtection": {
    "Type": "Boolean",
    "Description": "Whether to enable deletion protection.\nDefault to False.",
    "AllowedValues": [
     "True",
     "true",
     "False",
     "false"
    ],
    "Default": false
   },
   "AutoPay": {
    "Type": "Boolean",
    "Description": "Automatic Payment. Default is false.",
```

```
    "AllowedValues": [
     "True",
     "true",
     "False",
     "false"
    ],
    "Default": false
   },
   "Name": {
    "Type": "String",
    "Description": "The name of the EIP. The name must be 2 to 128 characters in le
ngth. It must start with a letter. It can contain numbers, periods (.), underscores
(_), and hyphens (-). It cannot start with http://  or https://"
   },
   "InternetChargeType": {
    "Type": "String",
    "Description": "The network charge type. Support 'PayByBandwidth' and 'PayByTra
ffic' only. Default is PayByBandwidth. PayByTraffic will charge by hour, PayByBandw
idth will charge by day. ",
    "AllowedValues": [
     "PayByBandwidth",
     "PayByTraffic"
    ],
    "Default": "PayByBandwidth"
   },
   "Netmode": {
    "Type": "String",
    "Description": "The network type. Valid value: public (public network)."
   },
   "Bandwidth": {
    "Type": "Number",
    "Description": "Bandwidth for the output network. Default is 5MB.",
    "Default": 5
   },
   "Tags": {
    "Type": "Json",
    "Description": "Tags to attach to eip. Max support 20 tags to add during create
eip. Each tag with two properties Key and Value, and Key is required.",
    "MaxLength": 20
   }
  },
  "ROSTemplateFormatVersion": "2015-09-01",
  "Outputs": {
   "AllocationId": {
    "Description": "ID that Aliyun assigns to represent the allocation of the addre
ss for use with VPC. Returned only for VPC elastic IP addresses.",
    "Value": {
     "Fn::GetAtt": [
      "ElasticIp",
      "AllocationId"
     ]
    }
   },
   "EipAddress": {
    "Description": "IP address of created EIP.",
```

```
      "Description": "IP address of created EIP.",
      "Value": {
       "Fn::GetAtt": [
        "ElasticIp",
        "EipAddress"
       ]
      }
     },
     "OrderId": {
      "Description": "Order ID of prepaid EIP instance.",
      "Value": {
       "Fn::GetAtt": [
        "ElasticIp",
        "OrderId"
       ]
      }
     }
    },
    "Resources": {
     "ElasticIp": {
      "Type": "ALIYUN::VPC::EIP",
      "Properties": {
       "Description": {
        "Ref": "Description"
       },
       "ResourceGroupId": {
        "Ref": "ResourceGroupId"
       },
       "InstanceChargeType": {
        "Ref": "InstanceChargeType"
       },
       "PricingCycle": {
        "Ref": "PricingCycle"
       },
       "Isp": {
        "Ref": "Isp"
       },
       "Period": {
        "Ref": "Period"
       },
       "DeletionProtection": {
        "Ref": "DeletionProtection"
       },
       "AutoPay": {
        "Ref": "AutoPay"
       },
       "Name": {
        "Ref": "Name"
       },
       "InternetChargeType": {
        "Ref": "InternetChargeType"
       },
       "Netmode": {
        "Ref": "Netmode"
       },
```

```
      "Bandwidth": {
       "Ref": "Bandwidth"
      },
      "Tags": {
       "Ref": "Tags"
      }
     }
    }
   }
  }
}
```

3. 调用GetTemplateSummary接口获取模板信息，查询资源导入标识符。
   i. 搜索GetTemplateSummary接口。
   ii. 选择**RegionId**，将**TemplateBody**设置为从步骤2中获取的ALIYUN::VPC::EIP资源的模板，然后单击**发起调用**。

   调用结果如下：

```
{
 "ResourceTypes": [
  "ALIYUN::VPC::EIP"
 ],
 "Description": "No description",
 "Parameters": [
  {
   "NoEcho": "false",
   "Type": "Boolean",
   "Description": "Whether to enable deletion protection.\nDefault to False.",
   "AllowedValues": [
    "True",
    "true",
    "False",
    "false"
   ],
   "Label": "DeletionProtection",
   "Default": false,
   "ParameterKey": "DeletionProtection"
  },
  {
   "NoEcho": "false",
   "Type": "String",
   "Description": "Optional. The description of the EIP. The description must be 2
to 256 characters in length. It must start with a letter. It cannot start with http
://  or https://.",
   "Label": "Description",
   "ParameterKey": "Description"
  },
  {
   "NoEcho": "false",
   "Type": "Json",
   "Description": "Tags to attach to eip. Max support 20 tags to add during create
eip. Each tag with two properties Key and Value, and Key is required.",
   "Label": "Tags",
```

```
    "MaxLength": 20,
    "ParameterKey": "Tags"
   },
   {
    "NoEcho": "false",
    "Type": "String",
    "Description": "ISP tag for finance cloud region. only for cn-hangzhou and cn-qi
ngdao region), if you are not finance cloud user, this value will be ignore.",
    "Label": "Isp",
    "ParameterKey": "Isp"
   },
   {
    "NoEcho": "false",
    "Type": "Number",
    "Description": "Prepaid time period. While choose by pay by month, it could be f
rom 1 to 9. While choose pay by year, it could be from 1 to 3.",
    "Label": "Period",
    "MinValue": 1,
    "MaxValue": 9,
    "Default": 1,
    "ParameterKey": "Period"
   },
   {
    "NoEcho": "false",
    "Type": "String",
    "Description": "Resource group id.",
    "Label": "ResourceGroupId",
    "ParameterKey": "ResourceGroupId"
   },
   {
    "NoEcho": "false",
    "Type": "Boolean",
    "Description": "Automatic Payment. Default is false.",
    "AllowedValues": [
     "True",
     "true",
     "False",
     "false"
    ],
    "Label": "AutoPay",
    "Default": false,
    "ParameterKey": "AutoPay"
   },
   {
    "NoEcho": "false",
    "Type": "String",
    "Description": "The resource charge type. Default value is Postpaid",
    "AllowedValues": [
     "Prepaid",
     "Postpaid"
    ],
    "Label": "InstanceChargeType",
    "Default": "Postpaid",
    "ParameterKey": "InstanceChargeType"
```

```
  },
  {
   "NoEcho": "false",
   "Type": "String",
   "Description": "Price cycle of the resource. This property has no default value.
If ChargeType is specified as Postpaid, this value will be ignore.",
   "AllowedValues": [
    "Month",
    "Year"
   ],
   "Label": "PricingCycle",
   "Default": "Month",
   "ParameterKey": "PricingCycle"
  },
  {
   "NoEcho": "false",
   "Type": "String",
   "Description": "The network charge type. Support 'PayByBandwidth' and 'PayByTraf
fic' only. Default is PayByBandwidth. PayByTraffic will charge by hour, PayByBandwi
dth will charge by day. ",
   "AllowedValues": [
    "PayByBandwidth",
    "PayByTraffic"
   ],
   "Label": "InternetChargeType",
   "Default": "PayByBandwidth",
   "ParameterKey": "InternetChargeType"
  },
  {
   "NoEcho": "false",
   "Type": "Number",
   "Description": "Bandwidth for the output network. Default is 5MB.",
   "Label": "Bandwidth",
   "Default": 5,
   "ParameterKey": "Bandwidth"
  },
  {
   "NoEcho": "false",
   "Type": "String",
   "Description": "The network type. Valid value: public (public network).",
   "Label": "Netmode",
   "ParameterKey": "Netmode"
  },
  {
   "NoEcho": "false",
   "Type": "String",
   "Description": "The name of the EIP. The name must be 2 to 128 characters in len
gth. It must start with a letter. It can contain numbers, periods (.), underscores
(_), and hyphens (-). It cannot start with http://  or https://",
   "Label": "Name",
   "ParameterKey": "Name"
  }
 ],
 "RequestId": "2AA4188A-15D8-4BB4-9C26-847ED8315D20",
 "Version": "2015-09-01"
```

```
  "Version": "2015-09-01",
  "Metadata": {},
  "ResourceIdentifierSummaries": [
   {
    "LogicalResourceIds": [
     "ElasticIp"
    ],
    "ResourceType": "ALIYUN::VPC::EIP",
    "ResourceIdentifiers": [
     "AllocationId"
    ]
   }
  ]
}
```

iii. 查看 `ResourceIdentifierSummaries` 中 `ResourceIdentifiers` 的取值，即为待导入资源的标识符属性。

例如：本示例中 `ResourceIdentifiers` 取值为 `AllocationId` ，表示ALIYUN::VPC::EIP资源的标识符属性为弹性公网IP的ID。

10.3. 使用现有资源创建资源栈

您可以通过资源导入的方式创建资源栈。本文以导入弹性公网IP（EIP）资源为例，为您介绍使用现有资源创建资源栈的操作方法。

前提条件

资源导入前，请提前完成如下事项：

1. 获取EIP资源的标识符属性。
   本示例中，获取到的EIP资源的标识符属性为AllocationId，即EIP的实例ID。具体操作，请参见获取待导入资源的标识符属性。
2. 获取EIP的实例ID。
   登录EIP控制台，获取要导入的EIP的实例ID。

操作步骤

1. 登录资源编排控制台。
2. 在左侧导航栏，单击**资源栈**。
3. 在页面左上角的地域下拉列表，选择资源栈的所在地域。

   > **说明** 请确保待导入资源与资源栈处于同一地域。

4. 在**资源栈列表**页面，单击**创建资源栈**，然后在下拉列表中选择**使用现有资源（资源导入）**。
5. 在**选择模板**页面的**指定模板**区域，选择**选择已有模板**，设置**模板录入方式**为**输入模板**，并在**模板内容**区域输入如下JSON格式的模板，然后单击**下一步**。

```
{
  "ROSTemplateFormatVersion": "2015-09-01",
  "Resources": {
    "Eip": {
      "Type": "ALIYUN::VPC::EIP",
      "DeletionPolicy": "Retain",
      "Properties": {
        "Bandwidth": 5
      }
    }
  },
  "Outputs": {
    "EipAddress": {
      "Value": {
        "Fn::GetAtt": [
          "Eip",
          "EipAddress"
        ]
      }
    },
    "AllocationId": {
      "Value": {
        "Fn::GetAtt": [
          "Eip",
          "AllocationId"
        ]
      }
    }
  }
}
```

说明 `DeletionPolicy`：取值为`Retain`，表示删除保护策略为保留资源。为防止资源被误删除，请务必设置该项。

6. 在**识别资源**页面，输入资源标识符值（例如：`eip-bp140qv3j25nsfaqd****`），单击**下一步**。

7. 在**配置模板参数**页面，配置**资源栈名称**和**更改集名称**，单击**下一步**。
8. 在**配置资源栈**页面，配置相关参数，单击**下一步**。

   本示例使用默认配置。更多信息，请参见创建资源栈。

9. 在**检查并确认**页面，单击**创建资源栈和导入更改集**。
10. 在**更改集**页签，单击更改集右侧**操作**列的**执行**，执行更改集，开始资源导入。

执行结果

在**资源栈信息**页签，如果**基本信息**区域的**状态**显示为**导入创建完成**，则资源导入成功。

后续步骤

您可以在**资源栈信息**页签，单击**偏差状态**右侧的**检测偏差**，检测导入资源的模板与实际模板的匹配情况。具体操作，请参见检测资源栈的偏差状态。

10.4. 将现有资源导入资源栈

如果您想将现有的资源添加到已经创建的资源栈时，您可以通过直接导入资源的方式完成。本文以导入弹性公网IP（EIP）资源到已有资源栈为例进行介绍。

前提条件

资源导入前，请提前完成如下事项：

1. 获取EIP资源的标识符属性。
   参考获取待导入资源的标识符属性获取EIP资源的标识符属性，您在编辑模板的时候会使用该属性。
   本示例中，获取到的EIP资源的标识符属性为AllocationId，即EIP的实例ID。
2. 获取EIP的实例ID。
   登录EIP控制台，获取要导入的EIP的实例ID。

操作步骤

1. 登录资源编排控制台。
2. 在左侧导航栏，单击**资源栈**。
3. 在页面左上角的地域下拉列表，选择资源栈的所在地域。

   > **说明** 请确保待导入资源与资源栈处于同一地域。

4. 在**资源栈列表**页面，在资源栈右侧**操作**列，选择 > **导入资源**。
5. 在**选择模板**页面，将**模板录入方式**设置为**输入模板**，在**模板内容**区域修改模板，增加待导入资源，然后单击**下一步**。

   本示例中，资源栈中已经存在一个EIP，新导入的资源我们命名为EIP2。示例模板如下所示：

```
{
  "ROSTemplateFormatVersion": "2015-09-01",
  "Resources": {
    "Eip": {
      "Type": "ALIYUN::VPC::EIP",
      "DeletionPolicy": "Retain",
      "Properties": {
        "Bandwidth": 5
      }
    },
    "Eip2": {
      "Type": "ALIYUN::VPC::EIP",
      "DeletionPolicy": "Retain",
      "Properties": {
        "Bandwidth": 5
      }
    }
  },
  "Outputs": {
    "EipAddress": {
      "Value": {
        "Fn::GetAtt": [
          "Eip",
          "EipAddress"
        ]
      }
    },
    "AllocationId": {
      "Value": {
        "Fn::GetAtt": [
          "Eip",
          "AllocationId"
        ]
      }
    },
    "EipAddress2": {
      "Value": {
        "Fn::GetAtt": [
          "Eip2",
          "EipAddress"
        ]
      }
    },
    "AllocationId2": {
      "Value": {
        "Fn::GetAtt": [
          "Eip2",
          "AllocationId"
        ]
      }
    }
  }
}
```

> **说明** `DeletionPolicy`：取值为`Retain`，表示删除保护策略为保留资源。为防止资源被误删除，请务必设置该项。

6. 在**识别资源**页面，输入资源标识符值（例如：`eip-bp1s1yz3aja40j377****`），单击**下一步**。

7. 在**配置模板参数**页面，配置**资源栈名称**和**更改集名称**，单击**下一步**。
8. 在**配置资源栈**页面，配置相关参数，单击**下一步**。

   本示例使用默认配置。更多信息，请参见创建资源栈。
9. 在**检查并确认**页面，单击**创建更改集**。
10. 单击更改集右侧**操作**列的**执行**，执行更改集，开始资源导入。

执行结果

在**资源**页签，查看EIP2资源是否已导入。

后续步骤

您可以在**资源栈信息**页签，单击**偏差状态**右侧的**检测偏差**，检测导入资源的模板与实际模板的匹配情况。具体操作，请参见检测资源栈的偏差状态。

10.5. 从资源栈移除资源

当您不需要资源栈中的某个资源时，您可以通过更新资源栈模板的方式移除该资源。本文以从资源栈中移除弹性公网IP（EIP）资源为例为您进行介绍。

背景信息

移除资源时，会面临两种情况：

- 从资源栈移除资源的同时，删除该资源本身。资源的删除保护`DeletionPolicy`设置为`Delete`。
- 仅从资源栈移除资源，保留资源本身。资源的删除保护`DeletionPolicy`设置为`Retain`。

本文提供的示例为第二种情况，即仅从资源栈移除资源，但要保留资源本身。

操作步骤

1. 登录资源编排控制台。
2. 在左侧导航栏，单击**资源栈**。
3. 在页面左上角的地域下拉列表，选择资源栈的所在地域。
4. 更新资源栈，将待移除资源（例如：EIP2）的 `DeletionPolicy` 设置为 `Retain` 。

   当EIP2的 `DeletionPolicy` 为 `Delete` 时需要执行该步骤，当 `DeletionPolicy` 为 `Retain` 时直接跳过该步骤。

   i. 在**资源栈列表**页面，单击资源栈右侧**操作**列的**更新**。

   ii. 单击**上一步**，然后在**选择模板**页面的**准备模板**区域，选择**替换当前模板**。

   iii. 在**模板录入方式**区域，选择**输入模板**，在**模板内容**区域修改EIP2的 `DeletionPolicy` 值为 `Retain` ，然后单击**下一步**。

   示例模板如下所示：

```
{
  "ROSTemplateFormatVersion": "2015-09-01",
  "Resources": {
    "Eip": {
      "Type": "ALIYUN::VPC::EIP",
      "Properties": {
        "Bandwidth": 5
      }
    },
    "Eip2": {
      "Type": "ALIYUN::VPC::EIP",
      "DeletionPolicy": "Retain",
      "Properties": {
        "Bandwidth": 5
      }
    }
  },
  "Outputs": {
    "EipAddress": {
      "Value": {
        "Fn::GetAtt": [
          "Eip",
          "EipAddress"
        ]
      }
    },
    "AllocationId": {
      "Value": {
        "Fn::GetAtt": [
          "Eip",
          "AllocationId"
        ]
      }
    },
    "EipAddress2": {
      "Value": {
        "Fn::GetAtt": [
          "Eip2",
          "EipAddress"
        ]
      }
    },
    "AllocationId2": {
      "Value": {
        "Fn::GetAtt": [
          "Eip2",
          "AllocationId"
        ]
      }
    }
  }
}
```

iv. 在**配置模板参数**页面，单击**确认修改**，完成资源栈更新。

5. 更新资源栈，移除EIP2资源。
   i. 在**资源栈列表**页面，单击资源栈右侧**操作**列的**更新**。
   ii. 单击**上一步**，然后在**选择模板**页面的**准备模板**区域，选择**替换当前模板**。
   iii. 在**模板录入方式**区域，选择**输入模板**，在**模板内容**区域修改模板内容，然后单击**下一步**。

   本示例中，需要在模板的 `Resources` 和 `Outputs` 中删除EIP2的参数。删除后的示例模板如下所示：

```
{
  "ROSTemplateFormatVersion": "2015-09-01",
  "Resources": {
    "Eip": {
      "Type": "ALIYUN::VPC::EIP",
      "Properties": {
        "Bandwidth": 5
      }
    }
  },
  "Outputs": {
    "EipAddress": {
      "Value": {
        "Fn::GetAtt": [
          "Eip",
          "EipAddress"
        ]
      }
    },
    "AllocationId": {
      "Value": {
        "Fn::GetAtt": [
          "Eip",
          "AllocationId"
        ]
      }
    }
  }
}
```

   iv. 在**配置模板参数**页面，单击**确认修改**，完成资源栈更新。

执行结果

移除资源成功后，资源栈中将不再包含EIP2资源。此时在资源栈的**资源**页签将不会看到EIP2资源信息，但您可以从VPC控制台的**弹性公网IP**页面查询到EIP2信息，表示EIP2资源仍然保留。

10.6. 在资源栈之间移动资源

您可以根据需要将资源从一个资源栈移动到另外一个资源栈。本文以移动弹性公网IP（EIP）资源为例进行介绍。

背景信息

本示例中，资源栈A中有一个EIP（例如：EIP2），您需要将EIP2移动到资源栈B中。

操作步骤

1. 将EIP2从资源栈A中移除。

   具体操作，请参见从资源栈移除资源。
   移除前，资源栈A的模板中包含EIP2。示例模板如下所示：

```
{
  "ROSTemplateFormatVersion": "2015-09-01",
  "Resources": {
    "Eip": {
      "Type": "ALIYUN::VPC::EIP",
      "Properties": {
        "Bandwidth": 5
      }
    },
    "Eip2": {
      "Type": "ALIYUN::VPC::EIP",
      "DeletionPolicy": "Retain",
      "Properties": {
        "Bandwidth": 5
      }
    }
  },
  "Outputs": {
    "EipAddress": {
      "Value": {
        "Fn::GetAtt": [
          "Eip",
          "EipAddress"
        ]
      }
    },
    "AllocationId": {
      "Value": {
        "Fn::GetAtt": [
          "Eip",
          "AllocationId"
        ]
      }
    },
    "EipAddress2": {
      "Value": {
        "Fn::GetAtt": [
          "Eip2",
          "EipAddress"
        ]
      }
    },
    "AllocationId2": {
      "Value": {
        "Fn::GetAtt": [
          "Eip2",
          "AllocationId"
        ]
      }
    }
  }
}
```

> **说明** `DeletionPolicy`：取值为`Retain`，表示删除保护策略为保留资源。为防止资源被误删除，请务必设置该项。

移除后，资源栈A的模板中不包含EIP2。示例模板如下所示：

```
{
  "ROSTemplateFormatVersion": "2015-09-01",
  "Resources": {
    "Eip": {
      "Type": "ALIYUN::VPC::EIP",
      "Properties": {
        "Bandwidth": 5
      }
    }
  },
  "Outputs": {
    "EipAddress": {
      "Value": {
        "Fn::GetAtt": [
          "Eip",
          "EipAddress"
        ]
      }
    },
    "AllocationId": {
      "Value": {
        "Fn::GetAtt": [
          "Eip",
          "AllocationId"
        ]
      }
    }
  }
}
```

2. 将EIP2导入资源栈B。

   具体操作，请参见将现有资源导入资源栈。
   导入前，资源栈B的模板中不包含EIP2。示例模板如下所示：

```
{
  "ROSTemplateFormatVersion": "2015-09-01"
}
```

   导入后，资源栈B的模板中包含EIP2。示例模板如下所示：

```
{
  "ROSTemplateFormatVersion": "2015-09-01",
  "Resources": {
    "Eip2": {
      "Type": "ALIYUN::VPC::EIP",
      "DeletionPolicy": "Retain",
      "Properties": {
        "Bandwidth": 5
      }
    }
  },
  "Outputs": {
    "EipAddress2": {
      "Value": {
        "Fn::GetAtt": [
          "Eip2",
          "EipAddress"
        ]
      }
    },
    "AllocationId2": {
      "Value": {
        "Fn::GetAtt": [
          "Eip2",
          "AllocationId"
        ]
      }
    }
  }
}
```

执行结果

操作成功后，EIP2将从资源栈A移动到资源栈B。您可以在资源栈B的**资源**页签，查看已导入的EIP2资源信息。

11.标签

11.1. 概览

资源编排服务ROS（Resource Orchestration Service）提供标签管理功能，方便您通过标签对ROS的资源栈或模板进行分类。
每个标签都由一对键值对组成，ROS标签的使用限制如下：

- 不支持未绑定资源栈或模板的空标签存在，标签必须绑定在某个资源栈或模板上。
- 一个资源栈或模板最多可以绑定20个标签。
- 一个资源栈或模板的每个标签的标签键必须唯一，相同标签键的标签会被覆盖。
- 每个地域中的标签信息不互通，例如在华东1（杭州）地域创建的标签在华东2（上海）地域不可见。

11.2. 使用标签管理资源栈

11.2.1. 创建标签

如果您的账号下有多个资源栈，可以通过给资源栈创建标签，方便您分类和统一管理。本文介绍如何在资源编排控制台创建标签。

操作步骤

1. 登录资源编排控制台。
2. 在左侧导航栏单击**资源栈**。
3. 在页面左上角的地域下拉列表，选择资源栈的所在地域。
4. 在**资源栈列表**页面，单击**创建资源栈**。
5. 在**选择模板**页面，根据需要选择模板，单击**下一步**。
6. 在**配置模板参数**页面，根据控制台提示，配置**资源栈名称**和**参数录入**，单击**下一步**。
7. 在**配置资源栈**页面，单击标签右侧的**添加**。

8. 在弹出的**编辑标签绑定**对话框，配置**标签键**和**标签值**，单击**确定**。

11.2.2. 使用标签搜索资源栈

为资源栈创建标签后，您可以使用标签快速搜索资源栈。

操作步骤

1. 登录资源编排控制台。
2. 在左侧导航栏单击**资源栈**。
3. 在页面左上角的地域下拉列表，选择资源栈的所在地域。
4. 在**资源栈列表**页面，单击**标签筛选**，选择**标签键**和**标签值**。

> **说明** 如果您未选择具体的标签值，搜索后会显示该标签键绑定的所有资源栈。

5. 单击**搜索**。

执行结果

资源栈列表页面将显示绑定该标签键和标签值的资源栈。

11.2.3. 删除标签

如果资源栈调整或者不再需要标签，您可以删除该资源栈的标签。

操作步骤

1. 登录资源编排控制台。
2. 在左侧导航栏单击**资源栈**。
3. 在页面左上角的地域下拉列表，选择资源栈的所在地域。
4. 在**资源栈列表**页面，找到需要删除标签的资源栈，单击**资源栈名称**列的资源栈ID。
5. 在**资源栈信息**页签，单击标签右侧的**编辑**。

6. 在弹出的**编辑标签绑定**对话框，单击标签右侧的

   。

7. 单击**确定**。

11.2.4. 标签传递

当您调用API创建或更新资源栈时，如果指定了标签，资源编排服务ROS（Resource Orchestration Service）会将标签传递到资源栈中每个支持标签的资源中。

使用说明

- 如果模板中定义的标签键和资源栈标签的标签键相同，则使用模板中定义的标签。例如：模板中资源A的标签是 `k1:v1` ，调用CreateStack创建资源栈时指定了 `k1:v2` ，则资源A最终的标签仍为 `k1:v1` 。
- 如果模板中未定义标签，或定义的标签键和资源栈标签键不同，则资源栈的标签会传递到模板定义的资源中。例如：模板中资源A的标签是 `k1:v1` ，调用CreateStack创建资源栈时指定了 `k2:v2` ，则资源A最终的标签为 `k1:v1,k2:v2` 。
- 只有当您调用CreateStack或UpdateStack创建或更新资源栈时，可以传递标签。调用TagResources为资源栈绑定标签时，不会传递标签。

支持标签传递的资源类型

云服务	资源类型
API网关	• ALIYUN::ApiGateway::Api • ALIYUN::ApiGateway::App • ALIYUN::ApiGateway::Group
内容分发网络CDN	ALIYUN::CDN::Domain
云企业网CEN	ALIYUN::CEN::CenInstance
云防火墙	ALIYUN::CLOUDFW::AddressBook
容器服务Kubernetes版ACK	• ALIYUN::CS::KubernetesCluster • ALIYUN::CS::ManagedEdgeKubernetesCluster • ALIYUN::CS::ManagedKubernetesCluster • ALIYUN::CS::ServerlessKubernetesCluster
弹性容器实例ECI	ALIYUN::ECI::ContainerGroup

云服务	资源类型
云解析DNS	ALIYUN::DNS::Domain
云原生分布式数据库PolarDB-X	ALIYUN::DRDS::DrdsInstance
云服务器ECS	• ALIYUN::ECS::AutoSnapshotPolicy • ALIYUN::ECS::CopyImage • ALIYUN::ECS::CustomImage • ALIYUN::ECS::DedicatedHost • ALIYUN::ECS::Disk • ALIYUN::ECS::Instance • ALIYUN::ECS::InstanceClone • ALIYUN::ECS::InstanceGroup • ALIYUN::ECS::InstanceGroupClone • ALIYUN::ECS::LaunchTemplate • ALIYUN::ECS::NetworkInterface • ALIYUN::ECS::SecurityGroup • ALIYUN::ECS::Snapshot • ALIYUN::ECS::SSHKeyPair • ALIYUN::ECS::VPC • ALIYUN::ECS::VSwitch
阿里云Elasticsearch	ALIYUN::ElasticSearch::Instance
弹性伸缩	ALIYUN::ESS::ScalingGroup
函数计算FC	ALIYUN::FC::Service
消息队列Kafka版	ALIYUN::KAFKA::Instance
云数据库MongoDB版	• ALIYUN::MONGODB::Instance • ALIYUN::MONGODB::ServerlessInstance • ALIYUN::MONGODB::ShardingInstance
文件存储NAS	ALIYUN::NAS::FileSystem
运维编排服务OOS	• ALIYUN::OOS::Execution • ALIYUN::OOS::Template
对象存储服务OSS	ALIYUN::OSS::Bucket
表格存储	ALIYUN::OTS::Instance
云数据库PolarDB	ALIYUN::POLARDB::DBCluster

云服务	资源类型
阿里云关系型数据库RDS	• ALIYUN::RDS::DBInstance • ALIYUN::RDS::DBInstanceClone • ALIYUN::RDS::PrepayDBInstance • ALIYUN::RDS::ReadOnlyDBInstance
云数据库Redis版	• ALIYUN::REDIS::Instance • ALIYUN::REDIS::PrepayInstance
消息队列RocketMQ版	ALIYUN::ROCKETMQ::Instance
Serverless应用引擎SAE	ALIYUN::SAE::Application
智能接入网关SAG	ALIYUN::SAG::CloudConnectNetwork
负载均衡SLB	• ALIYUN::SLB::AccessControl • ALIYUN::SLB::Certificate • ALIYUN::SLB::LoadBalancer • ALIYUN::SLB::LoadBalancerClone
日志服务SLS	ALIYUN::SLS::Project
专有网络VPC	• ALIYUN::VPC::CommonBandwidthPackage • ALIYUN::VPC::EIP • ALIYUN::VPC::Ipv6Gateway • ALIYUN::VPC::Ipv6InternetBandwidth • ALIYUN::VPC::NatGateway • ALIYUN::VPC::RouteTable • ALIYUN::VPC::VpnGateway

相关文档

使用资源编排为云资源批量绑定或更新标签

11.3. 使用标签管理模板

11.3.1. 创建标签

如果您的账号下有多个模板，可以通过给模板创建标签，方便您分类和统一管理。本文介绍如何在资源编排控制台创建标签。

操作步骤

1. 登录资源编排控制台。
2. 在左侧导航栏，选择**模板>我的模板**。
3. 在**我的模板**页面，单击**创建模板**。

4. 在弹出的**创建模板**对话框，单击标签右侧的**添加**。

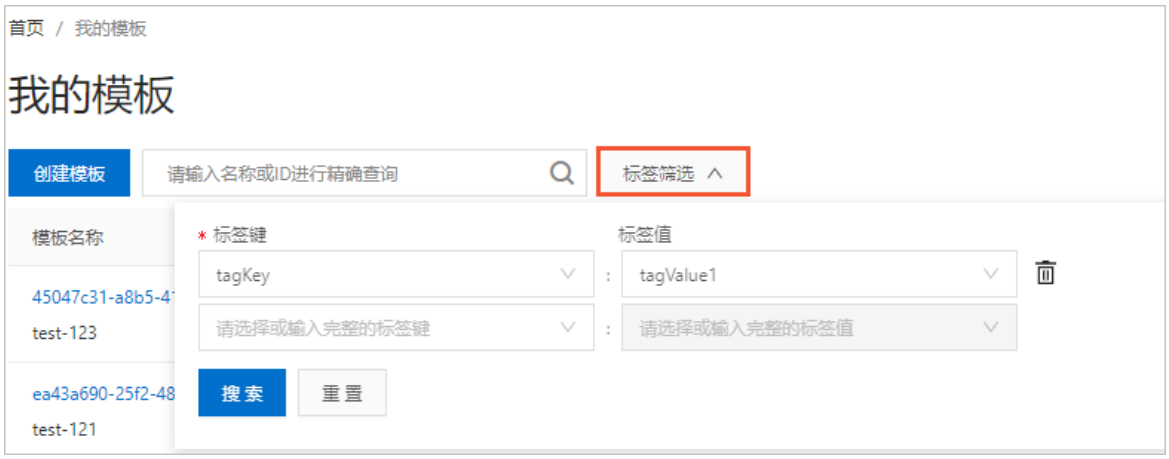
5. 在弹出的**编辑标签绑定**对话框，配置**标签键**和**标签值**，单击**确定**。

11.3.2. 使用标签搜索模板

为模板创建标签后，您可以使用标签快速搜索模板。

操作步骤

1. 登录资源编排控制台。
2. 在左侧导航栏，选择**模板>我的模板**。
3. 在**我的模板**页面，单击**标签筛选**，选择**标签键**和**标签值**。



> **说明** 如果您未选择具体的标签值，搜索后会显示该标签键绑定的模板。

4. 单击**搜索**。

执行结果

我的模板页面将显示绑定该标签键和标签值的模板。

11.3.3. 删除标签

如果模板调整或者不再需要标签，您可以删除该模板的标签。

操作步骤

1.
2.
3. 在**我的模板**页面找到需要删除标签的模板，单击右侧操作列的**编辑**。
4. 在弹出的**修改模板**对话框，单击标签右侧的**编辑**。
5. 在弹出的**编辑标签绑定**对话框，单击标签右侧的

   。

6. 单击**确定**。

12.安全与监控

12.1. 资源编排支持被审计的事件说明

资源编排已与操作审计服务集成，您可以在操作审计中查询用户操作资源编排产生的管控事件。操作审计支持将管控事件投递到日志服务SLS的LogStore或对象存储OSS的存储空间中，满足实时审计、问题回溯分析等需求。

操作审计记录了用户通过OpenAPI或控制台等方式操作云资源时产生的管控事件，资源编排支持在操作审计中查询的管控事件如下表所示。

事件名称	事件含义
CancelUpdateStack	取消更新资源栈。
ContinueCreateStack	资源栈创建失败后，重新创建资源栈。
CreateChangeSet	创建更改集。
CreateStack	创建资源栈。
CreateStackGroup	创建资源栈组。
CreateStackInstances	在指定账号和地域下创建资源栈实例。
CreateTemplate	创建自定义模板。
CreateTemplateScratch	创建资源场景。
DeleteChangeSet	删除更改集。
DeleteStack	删除资源栈。
DeleteStackGroup	删除资源栈组。
DeleteStackInstances	删除特定账号和地域下的资源栈实例。
DeleteTemplate	删除模板。
DeleteTemplateScratch	删除资源场景。
DescribeRegions	查询地域列表。
DetectStackDrift	对资源栈进行偏差检测。
DetectStackGroupDrift	对资源栈组进行偏差检测。
DetectStackResourceDrift	对资源进行偏差检测。
ExecuteChangeSet	执行更改集。
GenerateTemplateByScratch	为资源场景生成模板。

事件名称	事件含义
GenerateTemplatePolicy	生成模板策略。
GetChangeSet	查询更改集信息。
GetFeatureDetails	获取特定功能的详情。
GetResourceType	查询资源类型的详细信息。
GetResourceTypeTemplate	根据资源类型查询资源的模板。
GetServiceProvisions	获取服务支持的地域。
GetStack	查询资源栈信息。
GetStackDriftDetectionStatus	查询偏差检测的状态。
GetStackGroup	查询指定资源栈组的信息。
GetStackGroupOperation	查询资源栈组操作的信息。
GetStackInstance	查询指定资源栈组关联的资源栈实例的详细信息。
GetStackPolicy	获取资源栈策略。
GetStackResource	查询某个资源栈的资源。
GetTemplate	查询资源栈、更改集、自定义模板的模板详情。
GetTemplateEstimateCost	查询模板中创建资源的预估价格。
GetTemplateParameterConstraints	获取模板参数约束。
GetTemplateScratch	获取资源场景详情。
GetTemplateSummary	获取新模板或者现有模板的信息。
ListChangeSets	查询更改集列表。
ListResourceTypes	查询支持的资源类型列表。
ListStackEvents	查询资源栈及栈内资源的事件。
ListStackGroupOperationResults	查询资源栈组操作结果列表。

事件名称	事件含义
ListStackGroupOperations	查询资源栈组操作列表。
ListStackGroups	查询资源栈组列表。
ListStackInstances	查询指定资源栈组关联的资源栈实例列表。
ListStackOperationRisks	查询删除资源栈操作可能涉及的高风险资源，并返回每个资源对应的风险原因。
ListStackResourceDrifts	查询资源栈的资源偏差详情。
ListStackResources	查询某个资源栈的资源列表。
ListStacks	查询资源栈列表。
ListTagKeys	查询标签键。
ListTagResources	查询一个或多个ROS资源已经绑定的标签。
ListTagValues	查询指定标签键对应的标签值。
ListTemplates	查询模板列表。
ListTemplateScratches	查询资源场景列表。
ListTemplateVersions	查询模板版本列表。
MoveResourceGroup	修改资源所属资源组。
PreviewStack	预览指定模板将要创建的资源栈信息。
SetDeletionProtection	修改资源栈的删除保护属性。
SetStackPolicy	设置资源栈策略。
SetTemplatePermission	设置模板权限。
SignalResource	发送信号。
StopStackGroupOperation	停止资源栈组操作。
TagResources	为指定的ROS资源列表创建并绑定标签。
UntagResources	为指定的ROS资源列表统一解绑并删除标签。
UpdateStack	更新资源栈。
UpdateStackGroup	更新资源栈组。
UpdateStackInstances	在特定账号和地域下更新资源栈实例。

事件名称	事件含义
UpdateStackTemplateByResources	修正资源栈模板，消除资源栈的偏差。
UpdateTemplate	更新模板。
UpdateTemplateScratch	更新资源场景。
ValidateTemplate	验证将要创建资源栈的模板。

13.ROS CDK

13.1. 概览

阿里云ROS CDK（Cloud Development Toolkit）是资源编排（ROS）提供的一种命令行工具，帮助您使用多种编程语言定义云资源。您无需使用繁琐的JSON或YAML模板语法，即可使用ROS CDK完成资源的创建和配置，实现自动化部署及运维。

使用场景

利用面向对象的高级抽象模式对云资源进行标准定义，从而快速构建云资源。

工具优势

- 代码开源：已经在GitHub上开源，方便您快速获取工具代码并进行安装。
- 支持多种编程语言：JavaScript、TypeScript、Java、Python和C#。
- 极简代码开发：定义基础数据结构和逻辑，帮助您使用少量代码实现复杂模板的资源编排效果。
- 提供多个示例：支持通过OpenAPI开发者门户生成、下载和调试多种编程语言和多种资源的示例。

使用示例

- TypeScript示例
- JavaScript示例
- Java示例
- Python示例
- C#示例

13.2. 安装ROS CDK

本文为您介绍如何安装ROS CDK。

前提条件

请确保Node.js和TypeScript满足以下版本要求：

- Node.js：10.23.0及以上。
- TypeScript：2.7及以上。

在Linux平台安装ROS CDK

以下示例将在CentOS 8.2 64位的系统上安装ROS CDK。

1. 执行以下命令，安装Node.js、npm、tsc、lerna。

```
# 由于ROS CDK使用TypeScript开发，因此需要安装相关软件包。
yum install -y nodejs
npm install typescript -g
npm install lerna -g
```

2. 执行以下命令，安装CLI。

```
npm install @alicloud/ros-cdk-cli -g
```

3. 执行以下命令，查看ROS CDK功能列表。

```
ros-cdk
```

执行命令后，输出以下内容：

```
Usage: ros-cdk COMMAND
Commands:
  ros-cdk init [TEMPLATE]         Create a new, empty CDK project from a
                                  template. Invoked without TEMPLATE, the app
                                  template will be used.
  ros-cdk list [STACKS..]         Lists all stacks in the app      [aliases: ls]
  ros-cdk synthesize [STACKS..]   Synthesizes and prints the ROS template for
                                  this stack                    [aliases: synth]
  ros-cdk deploy [STACKS..]       Deploys the stack(s) named STACKS to ROS into
                                  your alicloud account
  ros-cdk diff [STACKS..]         Compares the specified stack with the deployed
                                  stack or a local template file, and returns
                                  with status 1 if any difference is found
  ros-cdk destroy [STACKS..]      Destroy the stack(s) named STACKS
  ros-cdk event [STACK..]         Get resource events within the resource STACK
  ros-cdk resource [STACKS..]     Get resources in the resource STACKS
  ros-cdk list-stacks [STACKS..]  Get resources in the resource STACKS
  ros-cdk load-config             Load Aliyun CLI config to CDK.
  ros-cdk config                  Set your alicloud account configuration.
Options:
  --json, -j       Use JSON output instead of YAML when templates are printed to
                   STDOUT                               [boolean] [default: false]
  --ignore-errors  Ignores synthesis errors, which will likely produce an
                   invalid output                       [boolean] [default: false]
  --trace          Print trace for stack warnings                       [boolean]
  --strict         Do not construct stacks with warnings                [boolean]
  --version        Show version number                                  [boolean]
  -h, --help       Show help                                            [boolean]
If your app has a single stack, there is no need to specify the stack name
If one of cdk.json or ~/.cdk.json exists, options specified there will be used
as defaults. Settings in cdk.json take precedence.
```

在Windows平台安装ROS CDK

以下示例将在Windows2016 64位的系统上安装ROS CDK。

1. 安装Node.js。
   i. 访问Node.js官网，下载Node.js安装包。
   ii. 据界面提示，安装Node.js。
   iii. 在cmd命令提示符窗口中执行以下命令，验证Node.js版本。

   ```
   node --version
   v10.23.0
   ```

2. 执行以下命令，安装npm、tsc、lerna。

   ```
   # 由于ROS CDK使用TypeScript开发，因此需要安装相关软件包。
   npm install typescript -g
   npm install lerna -g
   ```

3. 执行以下命令，安装CLI。

```
npm install @alicloud/ros-cdk-cli -g
```

4. 执行以下命令，查看ROS CDK功能列表。

```
ros-cdk
```

执行命令后，输出以下内容：

```
Usage: ros-cdk COMMAND
Commands:
  ros-cdk init [TEMPLATE]         Create a new, empty CDK project from a
                                  template. Invoked without TEMPLATE, the app
                                  template will be used.
  ros-cdk list [STACKS..]         Lists all stacks in the app      [aliases: ls]
  ros-cdk synthesize [STACKS..]   Synthesizes and prints the ROS template for
                                  this stack                    [aliases: synth]
  ros-cdk deploy [STACKS..]       Deploys the stack(s) named STACKS to ROS into
                                  your alicloud account
  ros-cdk diff [STACKS..]         Compares the specified stack with the deployed
                                  stack or a local template file, and returns
                                  with status 1 if any difference is found
  ros-cdk destroy [STACKS..]      Destroy the stack(s) named STACKS
  ros-cdk event [STACK..]         Get resource events within the resource STACK
  ros-cdk resource [STACKS..]     Get resources in the resource STACKS
  ros-cdk list-stacks [STACKS..]  Get resources in the resource STACKS
  ros-cdk load-config             Load Aliyun CLI config to CDK.
  ros-cdk config                  Set your alicloud account configuration.
Options:
  --json, -j       Use JSON output instead of YAML when templates are printed to
                   STDOUT                              [boolean] [default: false]
  --ignore-errors  Ignores synthesis errors, which will likely produce an
                   invalid output                      [boolean] [default: false]
  --trace          Print trace for stack warnings                      [boolean]
  --strict         Do not construct stacks with warnings               [boolean]
  --version        Show version number                                 [boolean]
  -h, --help       Show help                                           [boolean]
If your app has a single stack, there is no need to specify the stack name
If one of cdk.json or ~/.cdk.json exists, options specified there will be used
as defaults. Settings in cdk.json take precedence.
```

在macOS平台安装ROS CDK

以下示例将在macOS 11.2.2 64位的系统上安装ROS CDK。

1. 执行以下命令，安装brew、Node.js、npm、tsc、lerna。

```
curl -LsSf http://github.com/mxcl/homebrew/tarball/master | sudo tar xvz -C/usr/local --strip 1
brew install nodejs
npm install typescript -g
npm install lerna -g
```

2. 执行以下命令，安装CLI。

```
npm install @alicloud/ros-cdk-cli -g
```

3. 执行以下命令，查看ROS CDK功能列表。

```
ros-cdk
```

执行命令后，输出以下内容：

```
Usage: ros-cdk COMMAND
命令:
  ros-cdk init [TEMPLATE]         Create a new, empty CDK project from a
                                  template. Invoked without TEMPLATE, the app
                                  template will be used.
  ros-cdk list [STACKS..]         Lists all stacks in the app      [aliases: ls]
  ros-cdk synthesize [STACKS..]   Synthesizes and prints the ROS template for
                                  this stack                    [aliases: synth]
  ros-cdk deploy [STACKS..]       Deploys the stack(s) named STACKS to ROS into
                                  your alicloud account
  ros-cdk diff [STACKS..]         Compares the specified stack with the deployed
                                  stack or a local template file, and returns
                                  with status 1 if any difference is found
  ros-cdk destroy [STACKS..]      Destroy the stack(s) named STACKS
  ros-cdk event [STACK..]         Get resource events within the resource STACK
  ros-cdk resource [STACKS..]     Get resources in the resource STACKS
  ros-cdk list-stacks [STACKS..]  Get resources in the resource STACKS
  ros-cdk load-config             Load Aliyun CLI config to CDK.
  ros-cdk config                  Set your alicloud account configuration.
选项:
  --json, -j       Use JSON output instead of YAML when templates are printed to
                   STDOUT                                  [布尔] [默认值: false]
  --ignore-errors  Ignores synthesis errors, which will likely produce an
                   invalid output                          [布尔] [默认值: false]
  --trace          Print trace for stack warnings                          [布尔]
  --strict         Do not construct stacks with warnings                   [布尔]
  --version        显示版本号                                              [布尔]
  -h, --help       显示帮助信息                                            [布尔]
If your app has a single stack, there is no need to specify the stack name
If one of cdk.json or ~/.cdk.json exists, options specified there will be used
as defaults. Settings in cdk.json take precedence.
```

13.3. ROS CDK命令简介

本文为您介绍ROS CDK命令的含义及使用方法。

了解ROS CDK命令

命令	说明
ros-cdk init	初始化ROS CDK工程。
ros-cdk config	交互式配置阿里云账号信息。
ros-cdk load-config	从阿里云CLI获取阿里云凭证信息。

命令	说明
ros-cdk list（ls）	列出工程中的所有资源栈。
ros-cdk synthesize（synth）	由代码生成资源栈模板。
ros-cdk deploy	通过ROS创建资源栈。
ros-cdk list-stacks	通过ROS查询资源栈列表。
ros-cdk event	通过ROS查询资源栈事件。
ros-cdk output	通过ROS查询资源栈的输出信息。
ros-cdk resource	通过ROS查询资源栈资源列表。
ros-cdk diff	对比模板的差异。
ros-cdk destroy	通过ROS删除资源栈。
ros-cdk generate-stack-file	同步在工程内生成 `stack.info.json` 文件。
ros-cdk config-set	非交互式配置阿里云账号信息。

使用ROS CDK命令

- ros-cdk init ❯
- ros-cdk config ❯
- ros-cdk load-config ❯
- ros-cdk list （ls） ❯
- ros-cdk synthesize（synth） ❯
- ros-cdk deploy ❯
- ros-cdk list-stacks ❯
- ros-cdk event ❯
- ros-cdk output ❯
- ros-cdk resource ❯
- ros-cdk diff ❯
- ros-cdk destroy ❯
- ros-cdk generate-stack-file ❯
- ros-cdk config-set ❯

13.4. ROS CDK功能简介

本文以TypeScript语言为例，为您介绍ROS CDK功能。

参数

ROS支持使用参数（Parameters）提高模板的灵活性和可复用性，您可以通过以下代码在ROS CDK中设置参数。

关于参数（Parameters）的更多信息，请参见概览。

```
import * as ros from '@alicloud/ros-cdk-core';
import * as kms from '@alicloud/ros-cdk-kms';
export class DemoStack extends ros.Stack {
  constructor(scope: ros.Construct, id: string, props?: ros.StackProps) {
    super(scope, id, props);
    new ros.RosInfo(this, ros.RosInfo.description, "This is the simple ros cdk app example.
");
    // The code that defines your stack goes here
    const pendingWindowInDays = new ros.RosParameter(this, 'PendingWindowInDays', {
      description: 'The waiting period, specified in number of days. During this period, yo
u can cancel the CMK in PendingDeletion status. After the waiting period expires, you canno
t cancel the deletion. The value must be between 7 and 30. Default value is 30.',
      minLength: 7,
      maxLength: 30,
      defaultValue: 7,
      type: ros.RosParameterType.NUMBER
    });
    const aliasName = new ros.RosParameter(this, 'AliasName', {
      description: 'The display name of the key. You can use the alias to call APIs such as
Encrypt, GenerateDataKey, and DescribeKey. - Not including the prefix, the minimum length o
f an alias is 1 and the maximum length is 255. - The prefix alias/ must be included.',
      defaultValue: "alias/demo",
      type: ros.RosParameterType.STRING
    });
    const keyUsage = new ros.RosParameter(this, 'KeyUsage', {
      description: 'The intended use of the CMK. Default value: ENCRYPT/DECRYPT.',
      defaultValue: "ENCRYPT/DECRYPT",
      type: ros.RosParameterType.STRING,
      allowedValues: [
          "ENCRYPT/DECRYPT",
          "SIGN/VERIFY"
      ]
    });
    const keyEnable = new ros.RosParameter(this, 'keyEnable', {
      description: 'Specifies whether the key is enabled. Defaults to true.',
      defaultValue: true,
      type: ros.RosParameterType.BOOLEAN,
      allowedValues: [
        true,
        false
      ]
    });
    const Key = new kms.Key(this, 'ROS-KMS-Key', {
      keyUsage: keyUsage,
      enable: keyEnable,
      pendingWindowInDays: pendingWindowInDays
    });
    const Alias = new kms.Alias(this, 'ROS-KMS-Alias', {
      keyId: Key.attrKeyId,
      aliasName: aliasName
    });
  }
}
```

您可以解析ROS CDK代码，生成如下ROS模板：

```
{
  "Description": "This is the simple ros cdk app example.",
  "Metadata": {
    "ALIYUN::ROS::Interface": {
      "TemplateTags": [
        "Create by ROS CDK"
      ]
    }
  },
  "ROSTemplateFormatVersion": "2015-09-01",
  "Parameters": {
    "PendingWindowInDays": {
      "Type": "Number",
      "Default": 7,
      "Description": "The waiting period, specified in number of days. During this period, you can cancel the CMK in PendingDeletion status. After the waiting period expires, you can not cancel the deletion. The value must be between 7 and 30. Default value is 30.",
      "MaxLength": 30,
      "MinLength": 7
    },
    "AliasName": {
      "Type": "String",
      "Default": "alias/demo",
      "Description": "The display name of the key. You can use the alias to call APIs such as Encrypt, GenerateDataKey, and DescribeKey. - Not including the prefix, the minimum length of an alias is 1 and the maximum length is 255. - The prefix alias/ must be included."
    },
    "KeyUsage": {
      "Type": "String",
      "Default": "ENCRYPT/DECRYPT",
      "AllowedValues": [
        "ENCRYPT/DECRYPT",
        "SIGN/VERIFY"
      ],
      "Description": "The intended use of the CMK. Default value: ENCRYPT/DECRYPT."
    },
    "keyEnable": {
      "Type": "Boolean",
      "Default": true,
      "AllowedValues": [
        true,
        false
      ],
      "Description": "Specifies whether the key is enabled. Defaults to true."
    }
  },
  "Resources": {
    "ROS-KMS-Key": {
      "Type": "ALIYUN::KMS::Key",
      "Properties": {
        "Enable": {
          "Ref": "keyEnable"
```

```
        },
        "KeyUsage": {
          "Ref": "KeyUsage"
        },
        "PendingWindowInDays": {
          "Ref": "PendingWindowInDays"
        }
      }
    },
    "ROS-KMS-Alias": {
      "Type": "ALIYUN::KMS::Alias",
      "Properties": {
        "AliasName": {
          "Ref": "AliasName"
        },
        "KeyId": {
          "Fn::GetAtt": [
            "ROS-KMS-Key",
            "KeyId"
          ]
        }
      }
    }
  }
}
```

函数

ROS支持使用函数（Functions）管理资源栈，您可以通过以下代码在ROS CDK中设置函数。

关于函数（Functions）的更多信息，请参见函数（Functions）。

```
import * as ros from '@alicloud/ros-cdk-core';
import * as ecs from '@alicloud/ros-cdk-ecs';
export class DemoStack extends ros.Stack {
    constructor(scope: ros.Construct, id: string, props?: ros.StackProps) {
        super(scope, id, props);
        new ros.RosInfo(this, ros.RosInfo.description, "This is the simple ros cdk app exam
ple.");
        // The code that defines your stack goes here
        const vpcName = new ros.RosParameter(this, 'VpcName', {
            type: ros.RosParameterType.STRING,
            defaultValue:"VPC1,VPC2,VPC3"
        });
        const vswName = new ros.RosParameter(this, 'VSWName', {
            type: ros.RosParameterType.JSON,
            defaultValue: {"VSWName": "VSW1"}
        });
        const vpc = new ecs.Vpc(this, 'ROS-VPC', {
            vpcName: ros.Fn.select(0, ros.Fn.split(',', vpcName.valueAsString)),
            cidrBlock: '10.0.0.0/8',
            enableIpv6: false,
            description: 'This is the description of VPC',
        });
        const vswitch = new ecs.VSwitch(this, 'ROS-VSwitch', {
            vpcId: vpc.getAtt('VpcId'),
            vSwitchName: ros.Fn.getJsonValue('VSWName', vswName.valueAsAny),
            cidrBlock: '10.0.0.0/20',
            zoneId: ros.Fn.select(0, ros.Fn.getAzs(ros.Fn.ref('ALIYUN::Region'))),
        });
        vswitch.addDependency(vpc);
        const SubResultOutput = new ros.RosOutput(this, 'result', {
            value: ros.Fn.sub('Var1: ${Var1}, Var2: ${Var2}, StackName: ${ALIYUN::StackName
}, Region: ${ALIYUN::Region}', {
                Var1: 'Var1Value',
                Var2: 'Var2Value',
            }),
            description: 'test Sub function result',
        });
        const maxResultOutput = new ros.RosOutput(this, 'maxVal', {
            value: ros.Fn.max([1, 777]),
            description: 'test max function',
        });
        const minResultOutput = new ros.RosOutput(this, 'minVal', {
            value: ros.Fn.min([1, 777]),
            description: 'test min function',
        });
        const addResultOutput = new ros.RosOutput(this, 'testFunction1', {
            value: ros.Fn.add([3.14, 8.4]),
            description: 'number type add',
        });
    }
}
```

您可以解析ROS CDK代码，生成如下ROS模板：

```
{
  "Description": "This is the simple ros cdk app example.",
  "Metadata": {
    "ALIYUN::ROS::Interface": {
      "TemplateTags": [
        "Create by ROS CDK"
      ]
    }
  },
  "ROSTemplateFormatVersion": "2015-09-01",
  "Parameters": {
    "VpcName": {
      "Type": "String",
      "Default": "VPC1,VPC2,VPC3"
    },
    "VSWName": {
      "Type": "Json",
      "Default": {
        "VSWName": "VSW1"
      }
    }
  },
  "Resources": {
    "ROS-VPC": {
      "Type": "ALIYUN::ECS::VPC",
      "Properties": {
        "CidrBlock": "10.0.0.0/8",
        "Description": "This is the description of VPC",
        "EnableIpv6": false,
        "VpcName": {
          "Fn::Select": [
            0,
            {
              "Fn::Split": [
                ",",
                {
                  "Ref": "VpcName"
                }
              ]
            }
          ]
        }
      }
    },
    "ROS-VSwitch": {
      "Type": "ALIYUN::ECS::VSwitch",
      "Properties": {
        "CidrBlock": "10.0.0.0/20",
        "VpcId": {
          "Fn::GetAtt": [
            "ROS-VPC",
            "VpcId"
          ]
        },
```

```
        "ZoneId": {
          "Fn::Select": [
            0,
            {
              "Fn::GetAZs": {
                "Ref": "ALIYUN::Region"
              }
            }
          ]
        },
        "VSwitchName": {
          "Fn::GetJsonValue": [
            "VSWName",
            {
              "Ref": "VSWName"
            }
          ]
        }
      },
      "DependsOn": [
        "ROS-VPC"
      ]
    }
  },
  "Outputs": {
    "result": {
      "Description": "test Sub function result",
      "Value": {
        "Fn::Sub": [
          "Var1: ${Var1}, Var2: ${Var2}, StackName: ${ALIYUN::StackName}, Region: ${ALIYUN:
:Region}",
          {
            "Var1": "Var1Value",
            "Var2": "Var2Value"
          }
        ]
      }
    },
    "maxVal": {
      "Description": "test max function",
      "Value": {
        "Fn::Max": [
          1,
          777
        ]
      }
    },
    "minVal": {
      "Description": "test min function",
      "Value": {
        "Fn::Min": [
          1,
          777
        ]
```

```
      }
    },
    "testFunction1": {
      "Description": "number type add",
      "Value": {
        "Fn::Add": [
          3.14,
          8.4
        ]
      }
    }
  }
}
```

条件

ROS支持在模板中定义条件（Conditions），根据您在创建或更新资源栈时，指定的输入参数值进行计算。在每个条件中，都可以引用其他条件、参数值或映射。您可以通过以下代码在ROS CDK中设置条件。

关于条件（Conditions）的更多信息，请参见条件（Conditions）。

```
import * as ros from '@alicloud/ros-cdk-core';
import * as rds from '@alicloud/ros-cdk-rds';
export class DemoStack extends ros.Stack {
    constructor(scope: ros.Construct, id: string, props?: ros.StackProps) {
        super(scope, id, props);
        new ros.RosInfo(this, ros.RosInfo.description, "This is the simple ros cdk app exam
ple.");
        // The code that defines your stack goes here
        const rdsDBInstanceEngineType = new ros.RosParameter(this, 'RdsDBInstanceEngineType
', {
            type: ros.RosParameterType.STRING,
            defaultValue: 'sqlserver',
            allowedValues: [
                'sqlserver',
                'mysql'
            ]
        });
        const rdsDBInstanceId = new ros.RosParameter(this, 'RdsDBInstanceId', {
            type: ros.RosParameterType.STRING,
        });
        const rdsDBName = new ros.RosParameter(this, 'RdsDBName', {
            type: ros.RosParameterType.STRING,
        });
        const rdsEngineTypeConditions = new ros.RosCondition(this, 'RdsEngineTypeConditions
', {
            expression: ros.Fn.conditionEquals('mysql', rdsDBInstanceEngineType.valueAsStri
ng),
        });
        const rdsDataBase = new rds.Database(this, 'Database', {
            dbInstanceId: rdsDBInstanceId.valueAsString,
            dbName: rdsDBName.valueAsString,
            characterSetName: ros.Fn.conditionIf(rdsEngineTypeConditions.node.id, 'utf8', '
Chinese_PRC_CI_AS')
        })
    }
}
```

您可以解析ROS CDK代码，生成如下ROS模板：

```
{
  "Description": "This is the simple ros cdk app example.",
  "Metadata": {
    "ALIYUN::ROS::Interface": {
      "TemplateTags": [
        "Create by ROS CDK"
      ]
    }
  },
  "ROSTemplateFormatVersion": "2015-09-01",
  "Parameters": {
    "RdsDBInstanceEngineType": {
      "Type": "String",
      "Default": "sqlserver",
```

```
      "AllowedValues": [
        "sqlserver",
        "mysql"
      ]
    },
    "RdsDBInstanceId": {
      "Type": "String"
    },
    "RdsDBName": {
      "Type": "String"
    }
  },
  "Conditions": {
    "RdsEngineTypeConditions": {
      "Fn::Equals": [
        "mysql",
        {
          "Ref": "RdsDBInstanceEngineType"
        }
      ]
    }
  },
  "Resources": {
    "Database": {
      "Type": "ALIYUN::RDS::Database",
      "Properties": {
        "CharacterSetName": {
          "Fn::If": [
            "RdsEngineTypeConditions",
            "utf8",
            "Chinese_PRC_CI_AS"
          ]
        },
        "DBInstanceId": {
          "Ref": "RdsDBInstanceId"
        },
        "DBName": {
          "Ref": "RdsDBName"
        }
      }
    }
  }
}
```

映射

ROS支持Key-Value形式的映射（Mappings）表，您可以指定Key从而获取映射表的Value。您可以通过以下代码在ROS CDK中设置映射。

关于映射（Mappings）的更多信息，请参见映射（Mappings）。

```
import * as ros from '@alicloud/ros-cdk-core';
import * as ecs from '@alicloud/ros-cdk-ecs';
export class DemoStack extends ros.Stack {
    constructor(scope: ros.Construct, id: string, props?: ros.StackProps) {
        super(scope, id, props);
        new ros.RosInfo(this, ros.RosInfo.description, "This is the simple ros cdk app exam
ple.");
        // The code that defines your stack goes here
        const region = new ros.RosParameter(this, 'region', {
            type: ros.RosParameterType.STRING,
            allowedValues: ['cn-hangzhou', 'cn-beijing'],
        });
        const vpcId = new ros.RosParameter(this, 'VpcId', {
            type: ros.RosParameterType.STRING,
        });
        const vSwitchId = new ros.RosParameter(this, 'VSwitchId', {
            type: ros.RosParameterType.STRING,
        });
        const securityGroupId = new ros.RosParameter(this, 'SecurityGroupId', {
            type: ros.RosParameterType.STRING,
        });
        const ecsImageInfo = new ros.RosMapping(this, 'ECSImageInfo', {
            mapping: {
                "cn-hangzhou": {
                    "32": "m-25l0rcfjo",
                    "64": "m-25l0rcfj1"
                },
                "cn-beijing": {
                    "32": "m-25l0rcfj2",
                    "64": "m-25l0rcfj3"
                }
            },
        });
        const ecsGroups = new ecs.InstanceGroup(this,'ros-cdk-test-ecs',{
            vpcId: vpcId.valueAsString,
            vSwitchId: vSwitchId.valueAsString,
            imageId: ecsImageInfo.findInMap(region.valueAsString, '64'),
            maxAmount: 1,
            securityGroupId: securityGroupId.valueAsString,
            instanceType:'ecs.c6.large',
            instanceName: 'test-ros-cdk-ecs',
        });
    }
}
```

您可以解析ROS CDK代码，生成如下ROS模板：

```
{
  "Description": "This is the simple ros cdk app example.",
  "Metadata": {
    "ALIYUN::ROS::Interface": {
      "TemplateTags": [
        "Create by ROS CDK"
```

```
      ]
    }
  },
  "ROSTemplateFormatVersion": "2015-09-01",
  "Parameters": {
    "region": {
      "Type": "String",
      "AllowedValues": [
        "cn-hangzhou",
        "cn-beijing"
      ]
    },
    "VpcId": {
      "Type": "String"
    },
    "VSwitchId": {
      "Type": "String"
    },
    "SecurityGroupId": {
      "Type": "String"
    }
  },
  "Mappings": {
    "ECSImageInfo": {
      "cn-hangzhou": {
        "32": "m-25l0rcfjo",
        "64": "m-25l0rcfj1"
      },
      "cn-beijing": {
        "32": "m-25l0rcfj2",
        "64": "m-25l0rcfj3"
      }
    }
  },
  "Resources": {
    "ros-cdk-test-ecs": {
      "Type": "ALIYUN::ECS::InstanceGroup",
      "Properties": {
        "ImageId": {
          "Fn::FindInMap": [
            "ECSImageInfo",
            {
              "Ref": "region"
            },
            "64"
          ]
        },
        "InstanceType": "ecs.c6.large",
        "MaxAmount": 1,
        "AllocatePublicIP": true,
        "AutoRenew": "False",
        "AutoRenewPeriod": 1,
        "InstanceChargeType": "PostPaid",
        "InstanceName": "test-ros-cdk-ecs",
        "InternetChargeType": "PayByTraffic"
```

```
          "InternetChargeType": "PayByTraffic",
          "InternetMaxBandwidthIn": 200,
          "InternetMaxBandwidthOut": 1,
          "IoOptimized": "optimized",
          "Period": 1,
          "PeriodUnit": "Month",
          "SecurityGroupId": {
            "Ref": "SecurityGroupId"
          },
          "SystemDiskCategory": "cloud_efficiency",
          "VpcId": {
            "Ref": "VpcId"
          },
          "VSwitchId": {
            "Ref": "VSwitchId"
          }
        }
      }
    }
}
```

输出

ROS支持在输出（Outputs）中定义在调用查询资源栈接口时返回的值。您可以通过以下代码在ROS CDK中设置输出。

关于输出（Outputs）的更多信息，请参见输出（Outputs）。

```
import * as ros from '@alicloud/ros-cdk-core';
import * as ecs from '@alicloud/ros-cdk-ecs';
export class DemoStack extends ros.Stack {
    constructor(scope: ros.Construct, id: string, props?: ros.StackProps) {
        super(scope, id, props);
        new ros.RosInfo(this, ros.RosInfo.description, "This is the simple ros cdk app exam
ple.");
        // The code that defines your stack goes here
        const vpc = new ecs.Vpc(this, 'vpc-from-ros-cdk', {
            vpcName: 'test-ros-cdk',
            cidrBlock: '10.0.0.0/8',
            description: 'This is ros cdk test'
        });
        const vpcId = new ros.RosOutput(this, 'vpcId', {
            value:  vpc.attrVpcId,
            description: 'VpcId',
        });
        const vRouterId = new ros.RosOutput(this, 'VRouterId', {
            value:  vpc.attrVRouterId,
            description: 'VRouterId',
        });
        const routeTableId = new ros.RosOutput(this, 'RouteTableId', {
            value:  vpc.attrRouteTableId,
            description: 'RouteTableId',
        });
    }
}
```

您可以解析ROS CDK代码，生成如下ROS模板：

```
{
  "Description": "This is the simple ros cdk app example.",
  "Metadata": {
    "ALIYUN::ROS::Interface": {
      "TemplateTags": [
        "Create by ROS CDK"
      ]
    }
  },
  "ROSTemplateFormatVersion": "2015-09-01",
  "Resources": {
    "vpc-from-ros-cdk": {
      "Type": "ALIYUN::ECS::VPC",
      "Properties": {
        "CidrBlock": "10.0.0.0/8",
        "Description": "This is ros cdk test",
        "EnableIpv6": false,
        "VpcName": "test-ros-cdk"
      }
    }
  },
  "Outputs": {
    "vpcId": {
      "Description": "VpcId",
      "Value": {
        "Fn::GetAtt": [
          "vpc-from-ros-cdk",
          "VpcId"
        ]
      }
    },
    "VRouterId": {
      "Description": "VRouterId",
      "Value": {
        "Fn::GetAtt": [
          "vpc-from-ros-cdk",
          "VRouterId"
        ]
      }
    },
    "RouteTableId": {
      "Description": "RouteTableId",
      "Value": {
        "Fn::GetAtt": [
          "vpc-from-ros-cdk",
          "RouteTableId"
        ]
      }
    }
  }
}
```

伪参数

ROS支持将伪参数（Pseudo parameters）作为模板的固定参数。它们可以和用户定义的参数一样被引用，其值在ROS运行时确定。您可以通过以下代码在ROS CDK中设置伪参数。

关于伪参数（Pseudo parameters）的更多信息，请参见伪参数（Pseudo parameters）。

```
import * as ros from '@alicloud/ros-cdk-core';
export class DemoStack extends ros.Stack {
  constructor(scope: ros.Construct, id: string, props?: ros.StackProps) {
    super(scope, id, props);
    new ros.RosInfo(this, ros.RosInfo.description, "This is the simple ros cdk app example.
");
    // The code that defines your stack goes here
    const StackName = new ros.RosOutput(this, 'StackName', {
      value:  ros.RosPseudo.stackName,
    });
    const accountId = new ros.RosOutput(this, 'accountId', {
      value:  ros.RosPseudo.accountId,
    });
    const region = new ros.RosOutput(this, 'region', {
      value:  ros.RosPseudo.region,
    });
    const stackId = new ros.RosOutput(this, 'stackId', {
      value:  ros.RosPseudo.stackId,
    });
    const tenantId = new ros.RosOutput(this, 'tenantId', {
      value:  ros.RosPseudo.tenantId,
    });
  }
}
```

您可以解析ROS CDK代码，生成如下ROS模板：

```
{
  "Description": "This is the simple ros cdk app example.",
  "Metadata": {
    "ALIYUN::ROS::Interface": {
      "TemplateTags": [
        "Create by ROS CDK"
      ]
    }
  },
  "ROSTemplateFormatVersion": "2015-09-01",
  "Outputs": {
    "StackName": {
      "Value": {
        "Ref": "ALIYUN::StackName"
      }
    },
    "accountId": {
      "Value": {
        "Ref": "ALIYUN::AccountId"
      }
    },
    "region": {
      "Value": {
        "Ref": "ALIYUN::Region"
      }
    },
    "stackId": {
      "Value": {
        "Ref": "ALIYUN::StackId"
      }
    },
    "tenantId": {
      "Value": {
        "Ref": "ALIYUN::TenantId"
      }
    }
  }
}
```

元数据

ROS支持使用元数据（Metadata）对Parameters中定义的参数进行分组，并且为每一组分别定义标签。您可以通过以下代码在ROS CDK中设置元数据。

关于元数据（Metadata）的更多信息，请参见元数据（Metadata）。

```
import * as ros from '@alicloud/ros-cdk-core';
export class DemoStack extends ros.Stack {
    constructor(scope: ros.Construct, id: string, props?: ros.StackProps) {
        super(scope, id, props);
        new ros.RosInfo(this, ros.RosInfo.description, "This is the simple ros cdk app exam
ple.");
        // The code that defines your stack goes here
        const vpcId = new ros.RosParameter(this, 'VpcId', {
            type: ros.RosParameterType.STRING,
        });
        const metaData = new ros.RosInfo(this, ros.RosInfo.metadata, {
            'ALIYUN::ROS::Interface': {
                'ParameterGroups': {
                    "Parameters": [
                        vpcId.node.id
                    ],
                    "Label": {
                        "default": {
                            "zh-cn": "基础资源配置",
                            "en": "Infrastructure Configuration"
                        }
                    }
                },
                'TemplateTags': ['Demo Template']
            }
        });
    }
}
```

您可以解析ROS CDK代码，生成如下ROS模板：

```
{
  "Description": "This is the simple ros cdk app example.",
  "Metadata": {
    "ALIYUN::ROS::Interface": {
      "ParameterGroups": {
        "Parameters": [
          "VpcId"
        ],
        "Label": {
          "default": {
            "zh-cn": "基础资源配置",
            "en": "Infrastructure Configuration"
          }
        }
      },
      "TemplateTags": [
        "Demo Template",
        "Create by ROS CDK"
      ]
    }
  },
  "ROSTemplateFormatVersion": "2015-09-01",
  "Parameters": {
    "VpcId": {
      "Type": "String"
    }
  }
}
```

标签

ROS支持为创建的资源绑定标签。您可以通过以下代码在ROS CDK中设置标签。

```
import * as ros from '@alicloud/ros-cdk-core';
import * as ecs from '@alicloud/ros-cdk-ecs';
export class DemoStack extends ros.Stack {
  constructor(scope: ros.Construct, id: string, props?: ros.StackProps) {
    super(scope, id, props);
    new ros.RosInfo(this, ros.RosInfo.description, "This is the simple ros cdk app example.
");
    // The code that defines your stack goes here
    new ecs.Vpc(this, 'vpc-from-ros-cdk', {
      vpcName: 'test-ros-cdk',
      cidrBlock: '10.0.0.0/8',
      description: 'This is ros cdk test'
    });
    this.tags.setTag("MySampleTag", "MyTagValue");
  }
}
```

您可以执行以下命令，查询资源栈绑定的标签详情。

```
ros-cdk list-stacks
```

返回结果如下：

```
 The Stacks list is:
 [
 {
  "Status": "CREATE_COMPLETE",
  "StackType": "ROS",
  "ResourceGroupId": "rg-acfm2xw4X5w****",
  "StatusReason": "Stack CREATE completed successfully",
  "CreateTime": "2021-12-27T11:09:22",
  "RegionId": "cn-beijing",
  "DisableRollback": false,
  "StackName": "DemoStack",
  "Tags": [
   {
    "Value": "rg-acfm2xw4X5w****",
    "Key": "acs:rm:rgId"
   },
   {
    "Value": "MyTagValue",
    "Key": "MySampleTag"
   }
  ],
  "TimeoutInMinutes": 20,
  "StackId": "7a2a9f34-ae88-41d0-aae0-51aeerE****"
 }
]
```

资源

ROS支持为模板中的资源设置DependsOn、Count和Condition属性，具体如下：

- DependsOn
  设置DependsOn属性，可以指定特定资源紧跟着另一个资源后创建。为某个资源添加DependsOn属性后，该资源仅在DependsOn属性中指定的资源之后创建。您可以通过以下代码在ROS CDK中设置DependsOn属性。

```
import * as ros from '@alicloud/ros-cdk-core';
import * as ecs from '@alicloud/ros-cdk-ecs';
export class DemoStack extends ros.Stack {
  constructor(scope: ros.Construct, id: string, props?: ros.StackProps) {
    super(scope, id, props);
    new ros.RosInfo(this, ros.RosInfo.description, "This is the simple ros cdk app exampl
e.");
    // The code that defines your stack goes here
    const vpc = new ecs.Vpc(this, 'ROS-VPC', {
      vpcName: 'VPC_Name',
      cidrBlock: '10.0.0.0/8',
      enableIpv6: false,
      description: 'This is the description of VPC',
    });
    const vswitch = new ecs.VSwitch(this, 'ROS-VSwitch', {
      vpcId: vpc.getAtt('VpcId'),
      zoneId: 'cn-beijing-h',
      vSwitchName: 'VSwitch_Name',
      cidrBlock: '10.0.0.0/20'
    });
    vswitch.addDependency(vpc);
  }
}
```

您可以解析ROS CDK代码，生成如下ROS模板：

```
{
  "Description": "This is the simple ros cdk app example.",
  "Metadata": {
    "ALIYUN::ROS::Interface": {
      "TemplateTags": [
        "Create by ROS CDK"
      ]
    }
  },
  "ROSTemplateFormatVersion": "2015-09-01",
  "Resources": {
    "ROS-VPC": {
      "Type": "ALIYUN::ECS::VPC",
      "Properties": {
        "CidrBlock": "10.0.0.0/8",
        "Description": "This is the description of VPC",
        "EnableIpv6": false,
        "VpcName": "VPC_Name"
      }
    },
    "ROS-VSwitch": {
      "Type": "ALIYUN::ECS::VSwitch",
      "Properties": {
        "CidrBlock": "10.0.0.0/20",
        "VpcId": {
          "Fn::GetAtt": [
            "ROS-VPC",
            "VpcId"
          ]
        },
        "ZoneId": "cn-beijing-h",
        "VSwitchName": "VSwitch_Name"
      },
      "DependsOn": [
        "ROS-VPC"
      ]
    }
  }
}
```

- Count
  为资源指定Count属性后，ROS会对模板进行预处理，把该资源展开成多个资源。 操作资源栈时使用处理后的模板。您可以通过以下代码在ROS CDK中设置Count属性。

```
import * as ros from '@alicloud/ros-cdk-core';
import * as ecs from '@alicloud/ros-cdk-ecs';
export class DemoStack extends ros.Stack {
    constructor(scope: ros.Construct, id: string, props?: ros.StackProps) {
        super(scope, id, props);
        new ros.RosInfo(this, ros.RosInfo.description, "This is the simple ros cdk app ex
ample.");
        // The code that defines your stack goes here
        const vpc = new ecs.Vpc(this, 'ROS-VPC', {
            cidrBlock: '10.0.0.0/8',
            enableIpv6: false,
        });
        vpc.addCount(2)
        const vsw =new ecs.VSwitch(this, `vsw-from-ros-cdk`, {
            vpcId: ros.Fn.select(ros.RosPseudo.index, vpc.attrVpcId),
            zoneId: ros.Fn.select(ros.RosPseudo.index, ros.Fn.getAzs(ros.RosPseudo.region
)),
            cidrBlock: '10.0.0.0/16',
        });
        vsw.addCount(2)
    }
}
```

您可以解析ROS CDK代码，生成如下ROS模板：

```
{
  "Description": "This is the simple ros cdk app example.",
  "Metadata": {
    "ALIYUN::ROS::Interface": {
      "TemplateTags": [
        "Create by ROS CDK"
      ]
    }
  },
  "ROSTemplateFormatVersion": "2015-09-01",
  "Resources": {
    "ROS-VPC": {
      "Type": "ALIYUN::ECS::VPC",
      "Properties": {
        "CidrBlock": "10.0.0.0/8",
        "EnableIpv6": false
      },
      "Count": 2
    },
    "vsw-from-ros-cdk": {
      "Type": "ALIYUN::ECS::VSwitch",
      "Properties": {
        "CidrBlock": "10.0.0.0/16",
        "VpcId": {
          "Fn::Select": [
            {
              "Ref": "ALIYUN::Index"
            },
            {
              "Fn::GetAtt": [
                "ROS-VPC",
                "VpcId"
              ]
            }
          ]
        },
        "ZoneId": {
          "Fn::Select": [
            {
              "Ref": "ALIYUN::Index"
            },
            {
              "Fn::GetAZs": {
                "Ref": "ALIYUN::Region"
              }
            }
          ]
        }
      },
      "Count": 2
    }
  }
}
```

- Condition
  使用Condition属性可以指定是否需要创建资源。只有Condition所指定的条件值为True时，才会创建此资源。您可以通过以下代码在ROS CDK中设置Condition属性。

```
import * as ros from '@alicloud/ros-cdk-core';
import * as ecs from '@alicloud/ros-cdk-ecs';
export class DemoStack extends ros.Stack {
    constructor(scope: ros.Construct, id: string, props?: ros.StackProps) {
        super(scope, id, props);
        new ros.RosInfo(this, ros.RosInfo.description, "This is the simple ros cdk app ex
ample.");
        // The code that defines your stack goes here
        const createVPC = new ros.RosParameter(this, 'CreateVPC', {
            type: ros.RosParameterType.BOOLEAN,
            defaultValue: true,
        });
        const isCreateVPC = new ros.RosCondition(this, 'IsCreateVPC', {
            expression: ros.Fn.conditionEquals(true, createVPC.valueAsBoolean),
        });
        const vpc = new ecs.Vpc(this, 'ROS-VPC', {
            vpcName: 'VPC_Name',
            cidrBlock: '10.0.0.0/8',
            enableIpv6: false,
            description: 'This is the description of VPC',
        });
        vpc.addCondition(isCreateVPC)
    }
}
```

  您可以解析ROS CDK代码，生成如下ROS模板：

```
{
  "Description": "This is the simple ros cdk app example.",
  "Metadata": {
    "ALIYUN::ROS::Interface": {
      "TemplateTags": [
        "Create by ROS CDK"
      ]
    }
  },
  "ROSTemplateFormatVersion": "2015-09-01",
  "Parameters": {
    "CreateVPC": {
      "Type": "Boolean",
      "Default": true
    }
  },
  "Conditions": {
    "IsCreateVPC": {
      "Fn::Equals": [
        true,
        {
          "Ref": "CreateVPC"
        }
      ]
    }
  },
  "Resources": {
    "ROS-VPC": {
      "Type": "ALIYUN::ECS::VPC",
      "Properties": {
        "CidrBlock": "10.0.0.0/8",
        "Description": "This is the description of VPC",
        "EnableIpv6": false,
        "VpcName": "VPC_Name"
      },
      "Condition": "IsCreateVPC"
    }
  }
}
```

13.5. 使用示例

13.5.1. TypeScript示例

本文以创建一个阿里云专有网络（VPC）实例为例，为您介绍如何在TypeScript语言环境使用ROS CDK。

步骤一：初始化工程

每个ROS CDK应用都要求创建在一个独立的工程目录下，且该应用需要使用独立工程目录中模块的依赖项。所以在创建应用之前，需要先创建一个工程目录并进行初始化。

执行以下命令，创建工程目录并初始化工程。

```
mkdir demo
cd demo
ros-cdk init --language=typescript --generate-only=true
```

步骤二：配置阿里云凭证信息

1. 执行以下命令，配置阿里云凭证信息。

   ```
   ros-cdk config
   ```

2. 根据界面提示输入配置信息。

   ```
   endpoint(optional, default:https://ros.aliyuncs.com):
   defaultRegionId(optional, default:cn-hangzhou):cn-beijing
   [1] AK
   [2] StsToken
   [3] RamRoleArn
   [4] EcsRamRole
   [0] CANCEL
   Authenticate mode [1...4 / 0]: 1
   accessKeyId:**********************
   accessKeySecret:******************************
    Your cdk configuration has been saved successfully!
   ```

   配置内容说明如下：

   - endpoint：ROS服务地址。默认值为https://ros.aliyuncs.com。
   - defaultRegionId：ROS资源栈部署的地域。默认值为cn-hangzhou。
   - Authenticate mode：鉴权方式。本示例的鉴权方式为AccessKey，您需要输入accessKeyId和accessKeySecret。

步骤三（可选）：预览工程结构

执行以下命令，预览工程结构。

```
tree .
```

执行命令后，输出以下内容：

```
.
├── bin
│   └── demo.ts
├── cdk.json
├── jest.config.js
├── lib
│   └── demo-stack.ts
├── package.json
├── README.md
├── test
│   └── demo.test.ts
└── tsconfig.json
3 directories, 8 files
```

工程结构说明如下：

- `bin/demo.ts`：入口文件。

  > **说明** 一个工程有且仅有一个应用。

  示例中创建了一个应用（类型为ros.App）和一个资源栈（类型为DemoStack，名称为DemoStack），并将资源栈添加到应用中。`demo.ts` 文件内容如下：

```
#!/usr/bin/env node
import * as ros from '@alicloud/ros-cdk-core';
import { DemoStack } from '../lib/demo-stack';
const app = new ros.App({outdir: './cdk.out'});
new DemoStack(app, 'DemoStack');
```

- `lib/demo-stack.ts`：资源栈的定义文件。
  您可以将资源添加到资源栈中，动态构建资源栈。初始化生成的代码中，只为资源栈添加了描述信息。`demo-stack.ts` 文件内容如下：

```
import * as ros from '@alicloud/ros-cdk-core';
export class DemoStack extends ros.Stack {
  constructor(scope: ros.Construct, id: string, props?: ros.StackProps) {
    super(scope, id, props);
    new ros.RosInfo(this, ros.RosInfo.description, "This is the simple ros cdk app exampl
e.");
    // The code that defines your stack goes here
  }
}
```

- `test/demo.test.ts`：单元测试文件。
  用于验证构建资源栈的逻辑是否符合预期。`demo.test.ts` 文件内容如下：

```
import { expect as expectCDK, matchTemplate, MatchStyle } from '@alicloud/ros-cdk-assert'
;
import * as ros from '@alicloud/ros-cdk-core';
import * as Demo from '../lib/demo-stack';
test('Stack with version.', () => {
  const app = new ros.App();
  // WHEN
  const stack = new Demo.DemoStack(app, 'MyTestStack');
  // THEN
  expectCDK(stack).to(
    matchTemplate(
      {
        ROSTemplateFormatVersion: '2015-09-01',
        Description: "This is the simple ros cdk app example."
      },
      MatchStyle.EXACT,
    ),
  );
});
```

步骤四：安装依赖

1. 修改 `package.json` 文件，添加ECS依赖包。

```
{
  "name": "demo",
  "version": "0.1.0",
  "bin": {
    "demo": "bin/demo.js"
  },
  "scripts": {
    "build": "tsc",
    "test": "jest"
  },
  "devDependencies": {
    "@types/jest": "^25.2.1",
    "@types/node": "10.17.5",
    "typescript": "^3.9.7",
    "jest": "^25.5.0",
    "ts-jest": "^25.3.1",
    "ts-node": "^8.1.0",
    "babel-jest": "^26.6.3",
    "@babel/core": "^7.12.9",
    "@babel/preset-env": "7.12.7",
    "@babel/preset-typescript": "^7.12.7",
    "@alicloud/ros-cdk-assert": "^1.0.0"
  },
  "dependencies": {
    "@alicloud/ros-cdk-core": "^1.0.0",
    "@alicloud/ros-cdk-ecs": "^1.0.0"
  }
}
```

2. 执行以下命令，安装依赖。

```
npm install
```

步骤五：创建资源

1. 修改 `lib/demo-stack.ts` 文件，添加VPC。

```
import * as ros from '@alicloud/ros-cdk-core';
import * as ecs from '@alicloud/ros-cdk-ecs';
export class DemoStack extends ros.Stack {
  constructor(scope: ros.Construct, id: string, props?: ros.StackProps) {
    super(scope, id, props);
    new ros.RosInfo(this, ros.RosInfo.description, "This is the simple ros cdk app exam
ple.");
    // The code that defines your stack goes here
    new ecs.Vpc(this, 'vpc-from-ros-cdk', {
      vpcName: 'test-ros-cdk',
      cidrBlock: '10.0.0.0/8',
      description: 'This is ros cdk test'
   });
  }
}
```

2. 执行以下命令，生成模板文件。

```
ros-cdk synth --json
```

执行命令后，输出以下内容：

```
{
  "Description": "This is the simple ros cdk app example.",
  "ROSTemplateFormatVersion": "2015-09-01",
  "Resources": {
    "vpc-from-ros-cdk": {
      "Type": "ALIYUN::ECS::VPC",
      "Properties": {
        "CidrBlock": "10.0.0.0/8",
        "Description": "This is ros cdk test",
        "EnableIpv6": false,
        "VpcName": "test-ros-cdk"
      }
    }
  }
}
```

步骤六：单元测试

1. 修改 `test/demo.test.ts` 文件，验证资源栈创建VPC的可行性。

```
import { expect as expectCDK, matchTemplate, MatchStyle } from '@alicloud/ros-cdk-asser
t';
import * as ros from '@alicloud/ros-cdk-core';
import * as Demo from '../lib/demo-stack';
test('Stack with version.', () => {
  const app = new ros.App();
  // WHEN
  const stack = new Demo.DemoStack(app, 'MyDemoStack');
  // THEN
  expectCDK(stack).to(
    matchTemplate(
      {
        "Description": "This is the simple ros cdk app example.",
        "ROSTemplateFormatVersion": "2015-09-01",
        "Resources": {
          "vpc-from-ros-cdk": {
            "Type": "ALIYUN::ECS::VPC",
            "Properties": {
              "CidrBlock": "10.0.0.0/8",
              "Description": "This is ros cdk test",
              "EnableIpv6": false,
              "VpcName": "test-ros-cdk"
            }
          }
        }
      },
      MatchStyle.EXACT,
    ),
  );
});
```

2. 执行以下命令，进行单元测试。

```
npm test
```

执行命令后，输出以下内容：

```
> demo@0.1.0 test /root/demo
> jest
 PASS  test/demo.test.ts
  ✓ Stack with version. (136ms)
Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        2.988s, estimated 3s
Ran all test suites.
```

步骤七：管理资源栈

您可以使用ROS CDK命令，创建、查询或删除资源栈。

- 创建资源栈
  执行以下命令，创建资源栈。

```
ros-cdk deploy
```

执行命令后，输出以下内容：

```
□ The deployment(create stack) has completed!
RequestedId: BC963C80-054D-41A0-87E7-001E0AB7B1BA
StackId: 0714be3a-0713-4b7d-b7aa-1564adef****
```

- 查询资源栈
  执行以下命令，查询资源栈。

```
ros-cdk list-stacks
```

执行命令后，输出以下内容：

```
 □ The Stacks list is:
 [
        {
                "Status": "CREATE_COMPLETE",
                "StackType": "ROS",
                "StatusReason": "Stack CREATE completed successfully",
                "CreateTime": "2021-01-26T12:58:05",
                "RegionId": "cn-beijing",
                "DisableRollback": false,
                "StackName": "DemoStack",
                "Tags": [],
                "StackId": "218f0db0-7992-4e70-a586-15d****",
                "TimeoutInMinutes": 20
        }
]
```

- 删除资源栈
  i. 执行以下命令，删除资源栈。

```
ros-cdk destroy
```

  ii. 根据界面提示，确认删除。

```
The following stack(s) will be destroyed(Only deployed stacks will be displayed).
DemoStack
Please confirm.(Y/N)
Y
```

执行命令后，输出以下内容：

```
 □ Deleted
RequestedId: 1BF864E1-7965-4148-A302-E6ABFF806641
```

13.5.2. JavaScript示例

本文以创建一个阿里云专有网络（VPC）实例为例，为您介绍如何在JavaScript语言环境使用ROS CDK。

步骤一：初始化工程

每个ROS CDK应用都要求创建在一个独立的工程目录下，且该应用需要使用独立工程目录中模块的依赖项。所以在创建应用之前，需要先创建一个工程目录并进行初始化。

执行以下命令，创建工程目录并初始化工程。

```
mkdir demo
cd demo
ros-cdk init --language=javascript --generate-only=true
```

步骤二：配置阿里云凭证信息

1. 执行以下命令，配置阿里云凭证信息。

   ```
   ros-cdk config
   ```

2. 根据界面提示输入配置信息。

   ```
   endpoint(optional, default:https://ros.aliyuncs.com):
   defaultRegionId(optional, default:cn-hangzhou):cn-beijing
   [1] AK
   [2] StsToken
   [3] RamRoleArn
   [4] EcsRamRole
   [0] CANCEL
   Authenticate mode [1...4 / 0]: 1
   accessKeyId:**********
   accessKeySecret:******************************
    Your cdk configuration has been saved successfully!
   ```

   配置内容说明如下：

   - endpoint：ROS服务地址。默认值为https://ros.aliyuncs.com。
   - defaultRegionId：ROS资源栈部署的地域。默认值为cn-hangzhou。
   - Authenticate mode：鉴权方式。本示例的鉴权方式为AccessKey，您需要输入accessKeyId和accessKeySecret。

步骤三：预览工程结构

执行以下命令，预览工程结构。

```
tree .
```

执行命令后，输出以下内容：

```
.
├── bin
│   └── demo.js
├── cdk.json
├── jest.config.js
├── lib
│   └── demo-stack.js
├── package.json
├── README.md
└── test
    └── demo.test.js
3 directories, 7 files
```

工程结构说明如下：

- `bin/demo.js`：入口文件。

  > **说明** 一个工程有且仅有一个应用。

  示例中创建了一个应用（类型为ros.App）和一个资源栈（类型为DemoStack，名称为DemoStack），并将资源栈添加到应用中。`demo.js` 文件内容如下：

```
#!/usr/bin/env node
const ros = require('@alicloud/ros-cdk-core');
const { DemoStack } = require('../lib/demo-stack');
const app = new ros.App();
new DemoStack(app, 'DemoStack');
app.synth();
```

- `lib/demo-stack.js`：资源栈的定义文件。
  您可以将资源添加到资源栈中，动态构建资源栈。初始化生成的代码中，只为资源栈添加了描述信息。`demo-stack.js` 文件内容如下：

```
const ros = require('@alicloud/ros-cdk-core');
class DemoStack extends ros.Stack {
  /**
   *
   * @param {ros.Construct} scope
   * @param {string} id
   * @param {ros.StackProps} props
   */
  constructor(scope, id, props) {
    super(scope, id, props);
    new ros.RosInfo(this, ros.RosInfo.description, "This is the simple ros cdk app exampl
e.");
    // The code that defines your stack goes here
  }
}
module.exports = { DemoStack }
```

- `test/demo.test.js`：单元测试文件。
  用于验证构建资源栈的逻辑是否符合预期。`demo.test.js` 文件内容如下：

```
const { expect, matchTemplate, MatchStyle } = require('@alicloud/ros-cdk-assert');
const ros = require('@alicloud/ros-cdk-core');
const Demo = require('../lib/demo-stack');
test('Stack with version.', () => {
    const app = new ros.App();
    // WHEN
    const stack = new Demo.DemoStack(app, 'MyTestStack');
    // THEN
    expect(stack).to(matchTemplate({
            ROSTemplateFormatVersion: '2015-09-01',
            Description: "This is the simple ros cdk app example."
        }, MatchStyle.EXACT))
});
```

步骤四：安装依赖

1. 修改 `package.json` 文件，添加ECS依赖包。

```
{
  "name": "demo",
  "version": "0.1.0",
  "bin": {
    "demo": "bin/demo.js"
  },
  "scripts": {
    "build": "echo \"The build step is not required when using JavaScript!\" && exit 0"
,
    "cdk": "cdk",
    "test": "jest"
  },
  "devDependencies": {
    "@types/jest": "^25.2.1",
    "@types/node": "10.17.5",
    "typescript": "^3.9.7",
    "jest": "^25.5.0",
    "ts-jest": "^25.3.1",
    "ts-node": "^8.1.0",
    "babel-jest": "^26.6.3",
    "@babel/core": "^7.12.9",
    "@babel/preset-env": "7.12.7",
    "@babel/preset-typescript": "^7.12.7",
    "@alicloud/ros-cdk-assert": "^1.0.0"
  },
  "dependencies": {
    "@alicloud/ros-cdk-core": "^1.0.0",
    "@alicloud/ros-cdk-ecs": "^1.0.0"
  }
}
```

2. 执行以下命令，安装依赖。

```
npm install
```

步骤五：创建资源

1. 修改 `demo-stack.js` 文件，创建VPC。

```
const ros = require('@alicloud/ros-cdk-core');
const ecs = require('@alicloud/ros-cdk-ecs');
class DemoStack extends ros.Stack {
  /**
   *
   * @param {ros.Construct} scope
   * @param {string} id
   * @param {ros.StackProps} props
   */
  constructor(scope, id, props) {
    super(scope, id, props);
    new ros.RosInfo(this, ros.RosInfo.description, "This is the simple ros cdk app exam
ple.");
    // The code that defines your stack goes here
    new ecs.Vpc(this, 'vpc-from-ros-cdk', {
      vpcName: 'test-ros-cdk-javascript',
      cidrBlock: '10.0.0.0/8',
      description: 'This is ros cdk test'
   });
  }
}
module.exports = { DemoStack }
```

2. 执行以下命令，生成模板文件。

```
ros-cdk synth --json
```

执行命令后，输出以下内容：

```
{
  "ROSTemplateFormatVersion": "2015-09-01",
  "Resources": {
    "VPC": {
      "Type": "ALIYUN::ECS::VPC",
      "Properties": {
        "CidrBlock": "10.0.0.0/8",
        "Description": "This is ros cdk test",
        "EnableIpv6": false,
        "VpcName": "test-ros-cdk-javascript"
      }
    }
  }
}
```

步骤六：单元测试

1. 修改 `demo.test.js` 文件，验证资源栈创建VPC的可行性。

```
const {expect, matchTemplate, MatchStyle} = require('@alicloud/ros-cdk-assert');
const ros = require('@alicloud/ros-cdk-core');
const Demo = require('../lib/demo-stack');
test('Stack with version.', () => {
    const app = new ros.App();
    // WHEN
    const stack = new Demo.DemoStack(app, 'MyTestStack');
    // THEN
    expect(stack).to(matchTemplate(
        {
            "Description": "This is the simple ros cdk app example.",
            "ROSTemplateFormatVersion": "2015-09-01",
            "Resources": {
                "vpc-from-ros-cdk": {
                    "Type": "ALIYUN::ECS::VPC",
                    "Properties": {
                        "CidrBlock": "10.0.0.0/8",
                        "Description": "This is ros cdk test",
                        "EnableIpv6": false,
                        "VpcName": "test-ros-cdk-javascript"
                    }
                }
            }
        }
        , MatchStyle.EXACT))
});
```

2. 执行以下命令，进行单元测试。

```
npm test
```

执行命令后，输出以下内容：

```
> demo@0.1.0 test /opt/demo
> jest
 PASS  test/demo.test.js
  ✓ Stack with version. (257ms)
Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        1.79s
Ran all test suites.
```

步骤七：管理资源栈

您可以使用ROS CDK命令，创建、查询或删除资源栈。

- 创建资源栈
  执行以下命令，创建资源栈。

```
ros-cdk deploy
```

执行命令后，输出以下内容：

```
 □  The deployment(create stack) has completed!
RequestedId: BC963C80-054D-41A0-87E7-001E0AB7B1BA
StackId: 218f0db0-7992-4e70-a586-15****
```

- 查询资源栈
  执行以下命令，查询资源栈。

  ```
  ros-cdk list-stacks
  ```

  执行命令后，输出以下内容：

  ```
  □ The Stacks list is:
  [
          {
                  "Status": "CREATE_COMPLETE",
                  "StackType": "ROS",
                  "StatusReason": "Stack CREATE completed successfully",
                  "CreateTime": "2021-01-26T12:58:05",
                  "RegionId": "cn-beijing",
                  "DisableRollback": false,
                  "StackName": "DemoStack",
                  "Tags": [],
                  "StackId": "218f0db0-7992-4e70-a586-54e****",
                  "TimeoutInMinutes": 20
          }
  ]
  ```

- 删除资源栈
  i. 执行以下命令，删除资源栈。

  ```
  ros-cdk destroy
  ```

  ii. 根据界面提示，确认删除。

  ```
  The following stack(s) will be destroyed(Only deployed stacks will be displayed).
  DemoStack
  Please confirm.(Y/N)
  Y
  ```

  执行命令后，输出以下内容：

  ```
  □ Deleted
  RequestedId: 1BF864E1-7965-4148-A302-E6ABFF806641
  ```

13.5.3. Java示例

本文以创建一个阿里云专有网络（VPC）实例为例，为您介绍如何在Java语言环境使用ROS CDK。

前提条件

请确保JDK和Maven满足以下版本要求：

- JDK：8及以上。
- Maven：3.6及以上。

步骤一：初始化工程

每个ROS CDK应用都要求创建在一个独立的工程目录下，且该应用需要使用独立工程目录中模块的依赖项。所以在创建应用之前，需要先创建一个工程目录并进行初始化。

执行以下命令，创建工程目录并初始化工程。

```
mkdir demo
cd demo
ros-cdk init --language=java --generate-only=true
```

步骤二：配置阿里云凭证信息

1. 执行以下命令，配置阿里云凭证信息。

   ```
   ros-cdk config
   ```

2. 根据界面提示输入配置信息。

   ```
   endpoint(optional, default:https://ros.aliyuncs.com):
   defaultRegionId(optional, default:cn-hangzhou):cn-beijing
   [1] AK
   [2] StsToken
   [3] RamRoleArn
   [4] EcsRamRole
   [0] CANCEL
   Authenticate mode [1...4 / 0]: 1
   accessKeyId:**********
   accessKeySecret:**********
    Your cdk configuration has been saved successfully!
   ```

   配置内容说明如下：

   - endpoint：ROS服务地址。默认值为https://ros.aliyuncs.com。
   - defaultRegionId：ROS资源栈部署的地域。默认值为cn-hangzhou。
   - Authenticate mode：鉴权方式。本示例的鉴权方式为AccessKey，您需要输入accessKeyId和accessKeySecret。

步骤三：预览工程结构

执行以下命令，预览工程结构。

```
tree .
```

执行命令后，输出以下内容：

```
.
├── cdk.json
├── pom.xml
├── README.md
└── src
    ├── main
    │   └── java
    │       └── com
    │           └── myorg
    │               ├── DemoApp.java
    │               └── DemoStack.java
    └── test
        └── java
            └── com
                └── myorg
                    └── DemoTest.java
9 directories, 6 files
```

工程结构说明如下：

- `DemoApp.java`：入口文件。

  > **说明** 一个工程有且仅有一个应用。

  示例中创建了一个应用（类型为App）和一个资源栈（类型为DemoStack，名称为DemoStack），并将资源栈添加到应用中。`DemoApp.java`文件内容如下：

  ```
  package com.myorg;
  import com.aliyun.ros.cdk.core.*;
  import java.util.Arrays;
  public class DemoApp {
      public static void main(final String[] args) {
          App app = new App();
          new DemoStack(app, "DemoStack");
          app.synth();
      }
  }
  ```

- `DemoStack.java`：资源栈的定义文件。
  您可以将资源添加到资源栈中，动态构建资源栈。初始化生成的代码中，只为资源栈添加了描述信息。`DemoStack.java`文件内容如下：

```
package com.myorg;
import com.aliyun.ros.cdk.core.*;
public class DemoStack extends Stack {
    public DemoStack(final Construct scope, final String id) {
        this(scope, id, null);
    }
    public DemoStack(final Construct scope, final String id, final StackProps props) {
        super(scope, id, props);
        // The code that defines your stack goes here
    }
}
```

- `DemoTest.java`：单元测试文件。
  用于验证构建资源栈的逻辑是否符合预期。`DemoTest.java`文件内容如下：

```
package com.myorg;
import com.aliyun.ros.cdk.core.*;
import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.databind.SerializationFeature;
import com.fasterxml.jackson.databind.node.ObjectNode;
import org.junit.Test;
import java.io.IOException;
import static org.junit.Assert.assertEquals;
public class DemoTest {
    private final static ObjectMapper JSON =
        new ObjectMapper().configure(SerializationFeature.INDENT_OUTPUT, true);
    @Test
    public void testStack() throws IOException {
        App app = new App();
        DemoStack stack = new DemoStack(app, "test");
        // synthesize the stack to a ROS template and compare against
        // a checked-in JSON file.
        JsonNode actual = JSON.valueToTree(app.synth().getStackArtifact(stack.getArtifact
Id()).getTemplate());
        ObjectNode expected = new ObjectMapper().createObjectNode();
        expected.put("ROSTemplateFormatVersion", "2015-09-01");
        assertEquals(expected, actual);
    }
}
```

步骤四：安装依赖

1. 修改`pom.xml`文件，添加ECS SDK依赖包。

```
<?xml version="1.0" encoding="UTF-8"?>
<project xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/
xsd/maven-4.0.0.xsd"
         xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XM
LSchema-instance">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.myorg</groupId>
    <artifactId>java_demo</artifactId>
    <version>0.1</version>
```

```
    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <cdk.version>1.0.3</cdk.version>
    </properties>
    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.8.1</version>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.codehaus.mojo</groupId>
                <artifactId>exec-maven-plugin</artifactId>
                <version>1.6.0</version>
                <configuration>
                    <mainClass>com.myorg.DemoApp</mainClass>
                </configuration>
            </plugin>
        </plugins>
    </build>
    <dependencies>
        <!-- AliCloud ROS Cloud Development Kit (ROS CDK) -->
        <dependency>
            <groupId>com.aliyun</groupId>
            <artifactId>ros-cdk-core</artifactId>
            <version>1.0.0</version>
        </dependency>
        <dependency>
            <groupId>com.aliyun</groupId>
            <artifactId>ros-cdk-ecs</artifactId>
            <version>1.0.0</version>
        </dependency>
        <!-- https://mvnrepository.com/artifact/junit/junit -->
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.12</version>
            <scope>test</scope>
        </dependency>
    </dependencies>
</project>
```

2. 执行以下命令，安装依赖。

```
mvn compile
```

步骤五：创建资源

1. 修改 `DemoStack.java` 文件，创建VPC。

```
package com.myorg;
import com.aliyun.ros.cdk.core.*;
import com.aliyun.ros.cdk.ecs.*;
public class DemoStack extends Stack {
    public DemoStack(final Construct scope, final String id) {
        this(scope, id, null);
    }
    public DemoStack(final Construct scope, final String id, final StackProps props) {
        super(scope, id, props);
        // The code that defines your stack goes here
        Vpc.Builder.create(this, "VPC").vpcName("TestJavaCDK").description("This is ros
java cdk test").
                cidrBlock("10.0.0.0/8").build();
    }
}
```

2. 执行以下命令，生成模板文件。

```
mvn compile
ros-cdk synth --json
```

执行命令后，输出以下内容：

```
{
  "ROSTemplateFormatVersion": "2015-09-01",
  "Resources": {
    "VPC": {
      "Type": "ALIYUN::ECS::VPC",
      "Properties": {
        "CidrBlock": "10.0.0.0/8",
        "Description": "This is ros java cdk test",
        "EnableIpv6": false,
        "VpcName": "TestJavaCDK"
      }
    }
  }
}
```

步骤六：单元测试

1. 修改 `DemoTest.java` 文件，验证资源栈创建VPC的可行性。

```
package com.myorg;
import com.aliyun.ros.cdk.core.*;
import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.databind.SerializationFeature;
import com.fasterxml.jackson.databind.node.ObjectNode;
import org.junit.Test;
import java.io.IOException;
import static org.junit.Assert.assertEquals;
public class DemoTest {
    private final static ObjectMapper JSON =
            new ObjectMapper().configure(SerializationFeature.INDENT_OUTPUT, true);
    @Test
    public void testStack() throws IOException {
        App app = new App();
        DemoStack stack = new DemoStack(app, "test");
        // synthesize the stack to a ROS template and compare against
        // a checked-in JSON file.
        JsonNode actual = JSON.valueToTree(app.synth().getStackArtifact(stack.getArtifa
ctId()).getTemplate());
        ObjectNode expected = new ObjectMapper().createObjectNode();
        expected.put("ROSTemplateFormatVersion", "2015-09-01");
        ObjectNode resources = expected.putObject("Resources");
        ObjectNode vpc = resources.putObject("VPC");
        vpc.put("Type","ALIYUN::ECS::VPC");
        ObjectNode properties = vpc.putObject("Properties");
        properties.put("CidrBlock","10.0.0.0/8");
        properties.put("Description","This is ros java cdk test");
        properties.put("EnableIpv6",false);
        properties.put("VpcName","TestJavaCDK");
        assertEquals(expected, actual);
    }
}
```

2. 执行以下命令，进行单元测试。

```
mvn test
```

执行命令后，输出以下内容：

```
-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running com.myorg.DemoTest
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.294 sec
Results :
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  9.158 s
[INFO] Finished at: 2021-01-28T20:00:59+08:00
[INFO] ------------------------------------------------------------------------
```

步骤七：管理资源栈

您可以使用ROS CDK命令，创建、查询或删除资源栈。

- 创建资源栈
  执行以下命令，创建资源栈。

  ```
  ros-cdk deploy
  ```

  执行命令后，输出以下内容：

  ```
   The deployment(create stack) has completed!
  RequestedId: BC963C80-054D-41A0-87E7-001E0AB7B1BA
  StackId: 218f0db0-7992-4e70-a586-12****
  ```

- 查询资源栈
  执行以下命令，查询资源栈。

  ```
  ros-cdk list-stacks
  ```

  执行命令后，输出以下内容：

  ```
   The Stacks list is:
  [
          {
                  "Status": "CREATE_COMPLETE",
                  "StackType": "ROS",
                  "StatusReason": "Stack CREATE completed successfully",
                  "CreateTime": "2021-01-26T12:58:05",
                  "RegionId": "cn-beijing",
                  "DisableRollback": false,
                  "StackName": "DemoStack",
                  "Tags": [],
                  "StackId": "218f0db0-7992-4e70-a586-15e****",
                  "TimeoutInMinutes": 20
          }
  ]
  ```

- 删除资源栈

  i. 执行以下命令，删除资源栈。

  ```
  ros-cdk destroy
  ```

  ii. 根据界面提示，确认删除。

  ```
  The following stack(s) will be destroyed(Only deployed stacks will be displayed).
  DemoStack
  Please confirm.(Y/N)
  Y
  ```

  执行命令后，输出以下内容：

  ```
   Deleted
  RequestedId: 1BF864E1-7965-4148-A302-E6ABFF806641
  ```

13.5.4. Python示例

本文为您介绍如何在Python语言环境通过ROS CDK创建一个阿里云专有网络（VPC）。

前提条件

请确保Python版本在3.6及以上。

步骤一：初始化工程

1. 执行以下命令，创建工程目录并初始化工程。

   每个ROS CDK应用都要求创建在一个独立的工程目录下，且该应用需要使用独立工程目录中模块的依赖项。所以在创建应用之前，需要先创建一个工程目录并进行初始化。

   ```
   mkdir demo
   cd demo
   ros-cdk init --language=python --generate-only=true
   ```

2. 执行以下命令，创建一个属于当前工程的虚拟环境。

   Python工程的运行依赖于虚拟环境（virtualenv），所以在初始化Python工程之后需要创建一个属于当前工程的虚拟环境。

   ```
   python3 -m venv .venv
   ```

3. 执行以下命令，进入虚拟环境。

   ```
   source .venv/bin/activate
   ```

步骤二：配置阿里云凭证信息

1. 执行以下命令，配置阿里云凭证信息。

   ```
   ros-cdk config
   ```

2. 根据界面提示输入配置信息。

   ```
   endpoint(optional, default:https://ros.aliyuncs.com):
   defaultRegionId(optional, default:cn-hangzhou):cn-beijing
   [1] AK
   [2] StsToken
   [3] RamRoleArn
   [4] EcsRamRole
   [0] CANCEL
   Authenticate mode [1...4 / 0]: 1
   accessKeyId:**********
   accessKeySecret:******************************
    Your cdk configuration has been saved successfully!
   ```

   配置内容说明如下：

   - endpoint：ROS服务地址。默认值为https://ros.aliyuncs.com。
   - defaultRegionId：ROS资源栈部署的地域。默认值为cn-hangzhou。
   - Authenticate mode：鉴权方式。本示例的鉴权方式为AccessKey，您需要输入AccessKey ID和AccessKey Secret。

步骤三（可选）：预览工程结构

执行以下命令，预览工程结构。

```
tree .
```

执行命令后，输出以下内容：

```
.
├── app.py
├── cdk.json
├── demo
│   ├── demo_stack.py
│   └── __init__.py
├── README.md
├── requirements.txt
├── setup.py
├── source.bat
└── test
    ├── __init__.py
    └── test_demo.py
2 directories, 10 files
```

工程结构说明如下：

- `app.py`：入口文件。

  > **说明** 一个工程有且仅有一个应用。

  示例中创建了一个应用（类型为core.App）和一个资源栈（类型为DemoStack，名称为demo），并将资源栈添加到应用中。`app.py` 文件内容如下：

  ```
  #!/usr/bin/env python3
  import ros_cdk_core as core
  from demo.demo_stack import DemoStack
  app = core.App()
  DemoStack(app, "demo")
  app.synth()
  ```

- `demo/demo_stack.py`：资源栈的定义文件。
  您可以将资源添加到资源栈中，动态构建资源栈。初始化生成的代码中，只为资源栈添加了描述信息。`demo_stack.py` 文件内容如下：

  ```
  import ros_cdk_core as core
  class DemoStack(core.Stack):
      def __init__(self, scope: core.Construct, construct_id: str, **kwargs) -> None:
          super().__init__(scope, construct_id, **kwargs)
          # The code that defines your stack goes here
  ```

- `test/test_demo.py`：单元测试文件。
  用于验证构建资源栈的逻辑是否符合预期。`test_demo.py` 文件内容如下：

```
#!/usr/bin/env python3
import unittest
import ros_cdk_core as core
from demo.demo_stack import DemoStack
class TestStack(unittest.TestCase):
    def setUp(self):
        pass
    def test_stack(self):
        app = core.App()
        stack = DemoStack(app, "testdemo")
        artifact = app.synth().get_stack_artifact(stack.artifact_id).template
        expect = {
                  "ROSTemplateFormatVersion": "2015-09-01"
                }
        self.assertDictEqual(artifact, expect)
    def tearDown(self):
        pass
if __name__ == '__main__':
    unittest.main()
```

步骤四：安装依赖

1. 修改 `requirements.txt` 文件，添加ECS SDK依赖包。

```
ros-cdk-core
ros-cdk-ecs
```

2. 执行以下命令，安装依赖。
   - 方法一：

```
pip install -r requirements.txt
```

   - 方法二：

```
pip install ros-cdk-core ros-cdk-ecs
```

步骤五：创建资源

1. 修改 `demo_stack.py` 文件，创建VPC。

```
import ros_cdk_core as core
import ros_cdk_ecs as ecs
class DemoStack(core.Stack):
    def __init__(self, scope: core.Construct, construct_id: str, **kwargs) -> None:
        super().__init__(scope, construct_id, **kwargs)
        # The code that defines your stack goes here
        props = {
                    'cidrBlock': '10.0.0.0/8',
                    'description': 'This is ros python cdk test',
                    'vpcName': 'test-ros-cdk-vpc'
                }
        ecs.Vpc(self, "VPC", props)
```

2. 执行以下命令，生成模板文件。

```
ros-cdk synth --json
```

执行命令后，输出以下内容：

```
{
  "ROSTemplateFormatVersion": "2015-09-01",
  "Resources": {
    "VPC": {
      "Type": "ALIYUN::ECS::VPC",
      "Properties": {
        "CidrBlock": "10.0.0.0/8",
        "Description": "This is ros python cdk test",
        "EnableIpv6": false,
        "VpcName": "test-ros-cdk-vpc"
      }
    }
  }
}
```

步骤六：单元测试

1. 修改 `test_demo.py` 文件，验证资源栈创建VPC的可行性。

```
#!/usr/bin/env python3
import unittest
import ros_cdk_core as core
from demo.demo_stack import DemoStack
class TestStack(unittest.TestCase):
    def setUp(self):
        pass
    def test_stack(self):
        app = core.App()
        stack = DemoStack(app, "testdemo")
        artifact = app.synth().get_stack_artifact(stack.artifact_id).template
        expect = {
                  "ROSTemplateFormatVersion": "2015-09-01",
                  "Resources": {
                    "VPC": {
                      "Type": "ALIYUN::ECS::VPC",
                      "Properties": {
                        "CidrBlock": "10.0.0.0/8",
                        "Description": "This is ros python cdk test",
                        "EnableIpv6": False,
                        "VpcName": "test-ros-cdk-vpc"
                      }
                    }
                  }
                }
        self.assertDictEqual(artifact, expect)
    def tearDown(self):
        pass
if __name__ == '__main__':
    unittest.main()
```

2. 执行以下命令，进行单元测试。

```
python -m unittest  -v
```

执行命令后，输出以下内容：

```
test_stack (test.test_demo.TestStack) ... ok
----------------------------------------------------------------------
Ran 1 test in 0.052s
OK
```

步骤七：管理资源栈

您可以使用ROS CDK命令，创建、查询或删除资源栈。

- 创建资源栈
  执行以下命令，创建资源栈。

```
ros-cdk deploy
```

执行命令后，输出以下内容：

```
 □  The deployment(create stack) has completed!
RequestedId: BC963C80-054D-41A0-87E7-001E0AB7B1BA
StackId: 218f0db0-7992-4e70-a586-12****
```

- 查询资源栈
  执行以下命令，查询资源栈。

```
ros-cdk list-stacks
```

执行命令后，输出以下内容：

```
 □ The Stacks list is:
 [
        {
                "Status": "CREATE_COMPLETE",
                "StackType": "ROS",
                "StatusReason": "Stack CREATE completed successfully",
                "CreateTime": "2021-01-26T12:58:05",
                "RegionId": "cn-beijing",
                "DisableRollback": false,
                "StackName": "DemoStack",
                "Tags": [],
                "StackId": "218f0db0-7992-4e70-a586-152es****",
                "TimeoutInMinutes": 20
        }
]
```

- 删除资源栈
  i. 执行以下命令，删除资源栈。

```
ros-cdk destroy
```

  ii. 根据界面提示，确认删除。

```
The following stack(s) will be destroyed(Only deployed stacks will be displayed).
DemoStack
Please confirm.(Y/N)
Y
```

执行命令后，输出以下内容：

```
□ Deleted
RequestedId: 1BF864E1-7965-4148-A302-E6ABFF806641
```

13.5.5. C#示例

本文以创建一个阿里云专有网络（VPC）实例为例，为您介绍如何在C#语言环境使用ROS CDK。

前提条件

请确保.NET Core版本在3.1及以上，并且.NET SDK版本在5.0及以上。

步骤一：初始化工程

每个ROS CDK应用都要求创建在一个独立的工程目录下，且该应用需要使用独立工程目录中模块的依赖项。所以在创建应用之前，需要先创建一个工程目录并进行初始化。

执行以下命令，创建工程目录并初始化工程。

```
mkdir demo
cd demo
ros-cdk init --language=csharp --generate-only=true
```

步骤二：配置阿里云凭证信息

1. 执行以下命令，配置阿里云凭证信息。

   ```
   ros-cdk config
   ```

2. 根据界面提示输入配置信息。

   ```
   endpoint(optional, default:https://ros.aliyuncs.com):
   defaultRegionId(optional, default:cn-hangzhou):cn-beijing
   [1] AK
   [2] StsToken
   [3] RamRoleArn
   [4] EcsRamRole
   [0] CANCEL
   Authenticate mode [1...4 / 0]: 1
   accessKeyId:**********************
   accessKeySecret:******************************
    □ Your cdk configuration has been saved successfully!
   ```

   配置内容说明如下：

   - endpoint：ROS服务地址。默认值为https://ros.aliyuncs.com。
   - defaultRegionId：ROS资源栈部署的地域。默认值为cn-hangzhou。
   - Authenticate mode：鉴权方式。本示例的鉴权方式为AccessKey，您需要输入AccessKey ID和AccessKey Secret。

步骤三：预览工程结构

执行以下命令，预览工程结构。

```
tree .
```

执行命令后，输出以下内容：

```
.
├── cdk.json
├── global.sln
├── README.md
└── src
    ├── Demo
    │   ├── Demo.csproj
    │   ├── DemoStack.cs
    │   ├── GlobalSuppressions.cs
    │   └── Program.cs
    └── DemoTest
        ├── DemoTest.cs
        └── DemoTest.csproj
3 directories, 9 files
```

初始化工程之后默认会定义一个应用文件（Program.cs），与一个资源栈文件（DemoStack.cs）。您可以在DemoStack.cs中通过自定义代码的方式动态构建资源栈。资源栈将被默认添加到应用中，每个工程只允许存在一个应用。工程结构说明如下：

- `DemoStack.cs`：资源栈文件。
  初始化生成的代码中，只为资源栈添加了描述信息。`DemoStack.cs`文件内容如下：

  ```
  using AlibabaCloud.SDK.ROS.CDK.Core;
  namespace Demo
  {
      public class DemoStack : Stack
      {
          public DemoStack(Construct scope, string id, IStackProps props = null) : base(sco
  pe, id, props)
          {
              // The code that defines your stack goes here
          }
      }
  }
  ```

- `Program.cs`：应用文件。
  您可以将资源栈添加到应用中。`Program.cs`文件内容如下：

```
using AlibabaCloud.SDK.ROS.CDK.Core;
using System;
using System.Collections.Generic;
using System.Linq;
namespace Demo
{
    sealed class Program
    {
        public static void Main(string[] args)
        {
            var app = new App();
            new DemoStack(app, "DemoStack");
            app.Synth();
        }
    }
}
```

- `DemoTest.cs`：单元测试文件。
  用于验证构建资源栈的逻辑是否符合预期。`DemoTest.cs` 文件内容如下：

```
using AlibabaCloud.SDK.ROS.CDK.Core;
using NUnit.Framework;
using Newtonsoft.Json;
using Newtonsoft.Json.Linq;
using Demo;
namespace Stack.UnitTests.Services
{
    [TestFixture]
    public class Stack_IsStackShould
    {
        [Test]
        public void DemoStack_IsStackShould()
        {
            var app = new App();
            var testStack = new DemoStack(app, "TestStack");
            var result = app.Synth().GetStackArtifact(testStack.ArtifactId).Template;
            var expectedJson = JsonConvert.SerializeObject(result);
            JObject obj = new JObject();
            obj.Add("ROSTemplateFormatVersion", "2015-09-01");
            var actual = JsonConvert.SerializeObject(obj);
            Assert.AreEqual(expectedJson, actual);
        }
    }
}
```

步骤四：安装依赖

修改 `Demo.csproj` 文件，添加ECS依赖包。

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>netcoreapp3.1</TargetFramework>
    <!-- Roll forward to future major versions of the netcoreapp as needed -->
    <RollForward>Major</RollForward>
  </PropertyGroup>
  <ItemGroup>
    <!-- CDK Construct Library dependencies -->
    <PackageReference Include="AlibabaCloud.SDK.ROS.CDK.Core" Version="1.0.0" />
    <PackageReference Include="AlibabaCloud.SDK.ROS.CDK.Ecs" Version="1.0.0" />
    <!-- jsii Roslyn analyzers (un-comment to obtain compile-time checks for missing requir
ed props
    <PackageReference Include="Amazon.Jsii.Analyzers" Version="*" PrivateAssets="all" />
    -->
  </ItemGroup>
</Project>
```

步骤五：创建资源

1. 修改 `DemoStack.cs` 文件，添加VPC。

```
using AlibabaCloud.SDK.ROS.CDK.Core;
using AlibabaCloud.SDK.ROS.CDK.Ecs;
namespace Demo
{
    public class DemoStack : Stack
    {
        public DemoStack(Construct scope, string id, IStackProps props = null) : base(s
cope, id, props)
        {
            new Vpc(this, "vpc-from-ros-cdk", new VPCProps
            {
                VpcName = "test-ros-cdk-csharp",
                CidrBlock = "10.0.0.0/8",
                Description = "This is ros cdk test"
            });
        }
    }
}
```

2. 执行以下命令，生成模板文件。

```
ros-cdk synth --json
```

执行命令后，输出以下内容：

```
{
  "ROSTemplateFormatVersion": "2015-09-01",
  "Resources": {
    "vpc-from-ros-cdk": {
      "Type": "ALIYUN::ECS::VPC",
      "Properties": {
        "CidrBlock": "10.0.0.0/8",
        "Description": "This is ros cdk test",
        "EnableIpv6": false,
        "VpcName": "test-ros-cdk-csharp"
      }
    }
  }
}
```

步骤六：单元测试

1. 修改 `DemoTest.cs` 文件，验证资源栈创建VPC的可行性。

```
using AlibabaCloud.SDK.ROS.CDK.Core;
using NUnit.Framework;
using Newtonsoft.Json;
using Newtonsoft.Json.Linq;
using Demo;
namespace Stack.UnitTests.Services
{
    [TestFixture]
    public class Stack_IsStackShould
    {
        [Test]
        public void DemoStack_IsStackShould()
        {
            var app = new App();
            var testStack = new DemoStack(app, "TestStack");
            var result = app.Synth().GetStackArtifact(testStack.ArtifactId).Template;
            var actualJson = JsonConvert.SerializeObject(result);
            JObject obj = new JObject();
            JObject Resources = new JObject();
            JObject VPC = new JObject();
            JObject Properties = new JObject();
            VPC.Add("Type", "ALIYUN::ECS::VPC");
            VPC.Add("Properties", Properties);
            Resources.Add("vpc-from-ros-cdk", VPC);
            Properties.Add("CidrBlock", "10.0.0.0/8");
            Properties.Add("Description", "This is ros cdk test");
            Properties.Add("EnableIpv6", false);
            Properties.Add("VpcName", "test-ros-cdk-csharp");
            obj.Add("ROSTemplateFormatVersion", "2015-09-01");
            obj.Add("Resources", Resources);
            var expected = JsonConvert.SerializeObject(obj);
            Assert.AreEqual(actualJson, expected);
        }
    }
}
```

2. 执行以下命令，进行单元测试。

```
dotnet test
```

执行命令后，输出以下内容：

```
Test run for /home/demo/src/DemoTest/bin/Debug/netcoreapp3.1/DemoTest.dll(.NETCoreApp,V
ersion=v3.1)
Microsoft (R) Test Execution Command Line Tool Version 16.7.1
Copyright (c) Microsoft Corporation.  All rights reserved.
Starting test execution, please wait...
A total of 1 test files matched the specified pattern.
Test Run Successful.
Total tests: 1
     Passed: 1
 Total time: 2.6478 Seconds
```

步骤七：管理资源栈

您可以使用ROS CDK命令，创建、查询或删除资源栈。

- 创建资源栈
  执行以下命令，创建资源栈。

  ```
  ros-cdk deploy
  ```

  执行命令后，输出以下内容：

  ```
   The deployment(create stack) has completed!
  RequestedId: FDFF3996-6CBC-48F9-A217-5FAEGXFE8
  StackId: 9c62f0f2-af85-4536-987a-154eg****
  ```

- 查询资源栈
  执行以下命令，查询资源栈。

  ```
  ros-cdk list-stacks
  ```

  执行命令后，输出以下内容：

  ```
   The Stacks list is:
  [
          {
                  "Status": "CREATE_COMPLETE",
                  "StackType": "ROS",
                  "StatusReason": "Stack CREATE completed successfully",
                  "CreateTime": "2021-02-19T03:30:30",
                  "RegionId": "cn-beijing",
                  "DisableRollback": false,
                  "StackName": "DemoStack",
                  "Tags": [],
                  "StackId": "9c62f0f2-af85-4536-987a-dgd2****",
                  "TimeoutInMinutes": 20
          }
  ]
  ```

- 删除资源栈
  i. 执行以下命令，删除资源栈。

  ```
  ros-cdk destroy
  ```

  ii. 根据界面提示，确认删除。

  ```
  The following stack(s) will be destroyed(Only deployed stacks will be displayed).
  DemoStack
  Please confirm.(Y/N)
  Y
  ```

  执行命令后，输出以下内容：

  ```
   Deleted
  RequestedId: 1BF864E1-7965-4148-A302-E6ABFF806641
  ```

13.6. 最佳实践

13.6.1. 使用ROS CDK和阿里云CLI管理资源栈

本文以TypeScript语言为例，为您介绍如何使用ROS CDK创建资源栈，然后使用阿里云CLI查询资源栈中的资源详情。

前提条件

请确保您已经完成以下操作：

- 安装和配置阿里云CLI。更多信息，请参见安装阿里云CLI和配置阿里云CLI。
- 安装ROS CDK。更多信息，请参见安装ROS CDK。
- 安装JSON解析工具包（工具包名称为 `jq` ）。更多信息，请参见JSON解析工具包。

步骤一：初始化工程

每个ROS CDK应用都要求创建在一个独立的工程目录下，且该应用需要使用独立工程目录中模块的依赖项。所以在创建应用之前，需要先创建一个工程目录并进行初始化。

执行以下命令，创建工程目录并初始化工程。

```
mkdir demo
cd demo
ros-cdk init --language=typescript --generate-only=true
```

步骤二：配置阿里云凭证信息

1. 执行以下命令，配置阿里云凭证信息。

   ```
   ros-cdk config
   ```

2. 根据界面提示输入配置信息。

   ```
   ros-cdk config
   endpoint(optional, default:https://ros.aliyuncs.com):
   defaultRegionId(optional, default:cn-hangzhou):cn-beijing
   [1] AK
   [2] StsToken
   [3] RamRoleArn
   [4] EcsRamRole
   [0] CANCEL
   Authenticate mode [1...4 / 0]: 1
   accessKeyId:************************
   accessKeySecret:******************************
    Your cdk configuration has been saved successfully!
   ```

步骤三：安装依赖

1. 修改 `package.json` 文件，添加ECS依赖包（ros-cdk-ecs）。

```
{
  "name": "demo",
  "version": "0.1.0",
  "bin": {
    "demo": "bin/demo.js"
  },
  "scripts": {
    "build": "tsc",
    "test": "jest"
  },
  "devDependencies": {
    "@types/jest": "^25.2.1",
    "@types/node": "10.17.5",
    "typescript": "^3.9.7",
    "jest": "^25.5.0",
    "ts-jest": "^25.3.1",
    "ts-node": "^8.1.0",
    "babel-jest": "^26.6.3",
    "@babel/core": "^7.12.9",
    "@babel/preset-env": "7.12.7",
    "@babel/preset-typescript": "^7.12.7",
    "@alicloud/ros-cdk-assert": "^1.0.0"
  },
  "dependencies": {
    "@alicloud/ros-cdk-core": "^1.0.5",
    "@alicloud/ros-cdk-ecs": "^1.0.3"
  }
}
```

2. 执行以下命令，安装依赖。

```
npm install
```

步骤四：创建资源栈

1. 修改 `lib/demo-stack.ts` 文件，创建资源栈。您可以在代码中定义VPC参数信息，从而创建VPC资源。

```
import * as ros from '@alicloud/ros-cdk-core';
import * as ecs from '@alicloud/ros-cdk-ecs';
export class DemoStack extends ros.Stack {
    constructor(scope: ros.Construct, id: string, props?: ros.StackProps) {
        super(scope, id, props);
        new ros.RosInfo(this, ros.RosInfo.description, "This is the simple ros cdk app
example.");
        // The code that defines your stack goes here
        const vpc = new ecs.Vpc(this, 'vpc-from-ros-cdk', {
            vpcName: 'test-ros-cdk',
            cidrBlock: '10.0.0.0/8',
            description: 'This is ros cdk test'
        });
        const vpcId = new ros.RosOutput(this, 'vpcId', {
            value:  vpc.attrVpcId,
        });
    }
}
```

2. 执行以下命令，生成模板文件。

```
ros-cdk synth --json
```

执行命令后，输出以下内容：

```
{
  "Description": "This is the simple ros cdk app example.",
  "Metadata": {
    "ALIYUN::ROS::Interface": {
      "TemplateTags": [
        "Create by ROS CDK"
      ]
    }
  },
  "ROSTemplateFormatVersion": "2015-09-01",
  "Resources": {
    "vpc-from-ros-cdk": {
      "Type": "ALIYUN::ECS::VPC",
      "Properties": {
        "CidrBlock": "10.0.0.0/8",
        "Description": "This is ros cdk test",
        "EnableIpv6": false,
        "VpcName": "test-ros-cdk"
      }
    }
  },
  "Outputs": {
    "vpcId": {
      "Value": {
        "Fn::GetAtt": [
          "vpc-from-ros-cdk",
          "VpcId"
        ]
      }
    }
  }
}
```

3. 执行以下命令，创建资源栈并生成 `stack.outputs.json` 文件。

```
ros-cdk deploy --sync=true --outputs-file=true
```

执行命令后，输出以下内容：

```
DemoStack: deploying...
|DemoStack               |2021-12-28T10:57:20 | CREATE_COMPLETE      | ALIYUN::ECS::VPC
| vpc-2zefn3x56bka624k6**** | vpc-from-ros-cdk
Outputs:
 Key: vpcId  Value: vpc-2zefn3x56bka624k6**** Description: No description given
 □ The deployment(sync deploy stack) has finished!
status: CREATE_COMPLETE
StatusReason: Stack CREATE completed successfully
StackId: 9dbd63f5-db09-4c16-b018-d3a3a7be****
```

4. 执行以下命令，查询资源栈的输出。

```
cat stack.outputs.json
```

执行命令后，输出以下内容：

```
{
    "DemoStack": [
        {
            "Description": "No description given",
            "OutputKey": "vpcId",
            "OutputValue": "vpc-2zefn3x56bka624ef****"
        }
    ]
}
```

步骤五：查询资源栈中的资源详情

使用阿里云CLI调用DescribeVpcAttribute接口，查询VPC实例详情。

```
vpcId=`cat stack.outputs.json |jq '.DemoStack[0].OutputValue'`| sed 's/\"//g'
aliyun vpc DescribeVpcAttribute --VpcId $vpcId
```

执行命令后，输出以下内容：

```
{
    "Description": "This is ros cdk test",
    "ClassicLinkEnabled": false,
    "ResourceGroupId": "rg-acfm2xwmef****",
    "SecondaryCidrBlocks": {
        "SecondaryCidrBlock": []
    },
    "CidrBlock": "10.0.0.0/8",
    "UserCidrs": {
        "UserCidr": []
    },
    "NetworkAclNum": 0,
    "VRouterId": "vrt-2zej8djrl78424efa****",
    "OwnerId": 1754580XXXXXX,
    "CloudResources": {
        "CloudResourceSetType": [
            {
                "ResourceCount": 1,
                "ResourceType": "VRouter"
            },
            {
                "ResourceCount": 1,
                "ResourceType": "RouteTable"
            }
        ]
    },
    "Status": "Available",
    "IsDefault": false,
    "RequestId": "3A742895-A93F-5F67-BA81-CCEE88EGAD324",
    "SupportIpv4Gateway": false,
    "Ipv4GatewayId": "",
    "VSwitchIds": {
        "VSwitchId": []
    },
    "VpcId": "vpc-2zefn3x56bka624ef****",
    "CreationTime": "2021-12-28T10:57:21Z",
    "VpcName": "test-ros-cdk",
    "RegionId": "cn-beijing",
    "Ipv6CidrBlock": ""
}
```


步骤六（可选）：删除资源栈

1. 执行以下命令，删除资源栈。

```
ros-cdk destroy --sync=true
```

2. 根据界面提示，确认删除。

```
The following stack(s) will be destroyed(Only deployed stacks will be displayed).
DemoStack
Please confirm.(Y/N)
y
```

执行命令后，输出以下内容：

```
DemoStack: destroying...
|DemoStack               | DELETE_COMPLETE      | ALIYUN::ECS::VPC        |  | vpc-from
-ros-cdk
 □ The task(sync destroy stack) has finished!
status: DELETE_COMPLETE
StatusReason: Stack DELETE completed successfully
StackId: 9dbd63f5-db09-4c16-b018-d3a3a7be****
```

13.6.2. 使用ROS CDK在同一个工程内切换应用管理资源栈

本文以TypeScript语言为例，为您介绍如何使用ROS CDK在同一个工程内切换应用管理资源栈。

步骤一：初始化工程

每个ROS CDK应用都要求创建在一个独立的工程目录下，且该应用需要使用独立工程目录中模块的依赖项。所以在创建应用之前，需要先创建一个工程目录并进行初始化。

执行以下命令，创建工程目录并初始化工程。

```
mkdir demo
cd demo
ros-cdk init --language=typescript --generate-only=true
```

步骤二：配置阿里云凭证信息

1. 执行以下命令，配置阿里云凭证信息。

   ```
   ros-cdk config
   ```

2. 根据界面提示输入配置信息。

   ```
   ros-cdk config
   endpoint(optional, default:https://ros.aliyuncs.com):
   defaultRegionId(optional, default:cn-hangzhou):cn-beijing
   [1] AK
   [2] StsToken
   [3] RamRoleArn
   [4] EcsRamRole
   [0] CANCEL
   Authenticate mode [1...4 / 0]: 1
   accessKeyId:**********************
   accessKeySecret:******************************
    □ Your cdk configuration has been saved successfully!
   ```

步骤三：安装依赖

1. 修改 `package.json` 文件，添加ECS依赖包（ros-cdk-ecs）和OSS依赖包（ros-cdk-oss）。

```
{
  "name": "demo",
  "version": "0.1.0",
  "bin": {
    "demo": "bin/demo.js"
  },
  "scripts": {
    "build": "tsc",
    "test": "jest"
  },
  "devDependencies": {
    "@types/jest": "^25.2.1",
    "@types/node": "10.17.5",
    "typescript": "^3.9.7",
    "jest": "^25.5.0",
    "ts-jest": "^25.3.1",
    "ts-node": "^8.1.0",
    "babel-jest": "^26.6.3",
    "@babel/core": "^7.12.9",
    "@babel/preset-env": "7.12.7",
    "@babel/preset-typescript": "^7.12.7",
    "@alicloud/ros-cdk-assert": "^1.0.0"
  },
  "dependencies": {
    "@alicloud/ros-cdk-core": "^1.0.5",
    "@alicloud/ros-cdk-ecs": "^1.0.3",
    "@alicloud/ros-cdk-oss": "^1.0.3"
  }
}
```

2. 执行以下命令，安装依赖。

```
npm install
```

步骤四：在同一个应用中管理多个资源栈

1. 修改 `lib/demo.ts` 文件，以便在同一个应用中管理多个资源栈（资源栈 `DemoStack` 和资源栈 `DemoStack2` ）。

```
#!/usr/bin/env node
import * as ros from '@alicloud/ros-cdk-core';
import { DemoStack } from '../lib/demo-stack';
import { DemoStack2 } from '../lib/demo-stack';
const app = new ros.App({outdir: './cdk.out'});
new DemoStack(app, 'DemoStack');
new DemoStack2(app, 'DemoStack2');
app.synth();
```

2. 修改 `lib/demo-stack.ts` 文件，在资源栈 `DemoStack` 中创建专有网络，在资源栈 `DemoStack2` 中创建对象存储OSS的存储空间。

```
import * as ros from '@alicloud/ros-cdk-core';
import * as ecs from '@alicloud/ros-cdk-ecs';
import * as oss from '@alicloud/ros-cdk-oss';
export class DemoStack extends ros.Stack {
    constructor(scope: ros.Construct, id: string, props?: ros.StackProps) {
        super(scope, id, props);
        new ros.RosInfo(this, ros.RosInfo.description, "This is the simple ros cdk app
example.");
        // The code that defines your stack goes here
        new ecs.Vpc(this, 'vpc-from-ros-cdk', {
            vpcName: 'test-ros-cdk',
            cidrBlock: '10.0.0.0/8',
            description: 'This is ros cdk test'
        });
    }
}
export class DemoStack2 extends ros.Stack {
    constructor(scope: ros.Construct, id: string, props?: ros.StackProps) {
        super(scope, id, props);
        new ros.RosInfo(this, ros.RosInfo.description, "This is the simple ros cdk app
example.");
        // The code that defines your stack goes here
        new oss.Bucket(this, 'oss-from-ros-cdk', {
            bucketName: 'oss-from-ros-cdk'
            }
        )
    }
}
```

3. 生成模板文件。
   - 为资源栈 `DemoStack` 生成模板文件
     执行以下命令：

     ```
     ros-cdk synth DemoStack -j
     ```

     执行命令后，输出以下内容：

```
{
  "Description": "This is the simple ros cdk app example.",
  "Metadata": {
    "ALIYUN::ROS::Interface": {
      "TemplateTags": [
        "Create by ROS CDK"
      ]
    }
  },
  "ROSTemplateFormatVersion": "2015-09-01",
  "Resources": {
    "vpc-from-ros-cdk": {
      "Type": "ALIYUN::ECS::VPC",
      "Properties": {
        "CidrBlock": "10.0.0.0/8",
        "Description": "This is ros cdk test",
        "EnableIpv6": false,
        "VpcName": "test-ros-cdk"
      }
    }
  }
}
```

- 为资源栈 `DemoStack2` 生成模板文件
  执行以下命令：

```
ros-cdk synth DemoStack2 -j
```

执行命令后，输出以下内容：

```
{
  "Description": "This is the simple ros cdk app example.",
  "Metadata": {
    "ALIYUN::ROS::Interface": {
      "TemplateTags": [
        "Create by ROS CDK"
      ]
    }
  },
  "ROSTemplateFormatVersion": "2015-09-01",
  "Resources": {
    "oss-from-ros-cdk": {
      "Type": "ALIYUN::OSS::Bucket",
      "Properties": {
        "BucketName": "oss-from-ros-cdk",
        "AccessControl": "private",
        "DeletionForce": false
      }
    }
  }
}
```

4. 创建资源栈。
   - 创建资源栈 `DemoStack`

执行以下命令：

```
ros-cdk deploy DemoStack --sync=true
```

执行命令后，输出以下内容：

```
DemoStack: deploying...
|DemoStack               |2021-12-29T06:56:00 | CREATE_COMPLETE      | ALIYUN::ECS::V
PC        | vpc-2ze18c8hsomme5ps**** | vpc-from-ros-cdk
 □ The deployment(sync deploy stack) has finished!
status: CREATE_COMPLETE
StatusReason: Stack CREATE completed successfully
StackId: e002686b-d269-41df-bdf4-27221b1****
```

- 创建资源栈 `DemoStack2`

  执行以下命令：

  ```
  ros-cdk deploy DemoStack2 --sync=true
  ```

  执行命令后，输出以下内容：

  ```
  DemoStack2: deploying...
  |DemoStack2              |2021-12-29T06:56:24 | CREATE_COMPLETE      | ALIYUN::OSS::B
  ucket     | oss-from-ros-cdk | oss-from-ros-cdk
   □ The deployment(sync deploy stack) has finished!
  status: CREATE_COMPLETE
  StatusReason: Stack CREATE completed successfully
  StackId: 502a2cbe-3ab2-49fa-bcaf-602c3b88****
  ```

5. （可选）删除资源栈。
   - 删除资源栈 `DemoStack`

     a. 执行以下命令：

     ```
     ros-cdk destroy DemoStack --sync=true
     ```

     b. 根据界面提示，确认删除。

     ```
     The following stack(s) will be destroyed(Only deployed stacks will be displayed).
     DemoStack
     Please confirm.(Y/N)
     y
     ```

     执行命令后，输出以下内容：

     ```
     DemoStack: destroying...
     |DemoStack               | DELETE_COMPLETE      | ALIYUN::ECS::VPC        |  | vp
     c-from-ros-cdk
      □ The task(sync destroy stack) has finished!
     status: DELETE_COMPLETE
     StatusReason: Stack DELETE completed successfully
     StackId: e002686b-d269-41df-bdf4-27221****
     ```

   - 删除资源栈 `DemoStack2`

a. 执行以下命令：

```
ros-cdk destroy DemoStack2 --sync=true
```

b. 根据界面提示，确认删除。

```
The following stack(s) will be destroyed(Only deployed stacks will be displayed).
DemoStack2
Please confirm.(Y/N)
y
```

执行命令后，输出以下内容：

```
DemoStack2: destroying...
|DemoStack2              | DELETE_COMPLETE      | ALIYUN::OSS::Bucket     |  | os
s-from-ros-cdk
 The task(sync destroy stack) has finished!
status: DELETE_COMPLETE
StatusReason: Stack DELETE completed successfully
StackId: 502a2cbe-3ab2-49fa-bcaf-602c3b88****
```

步骤五：切换应用管理资源栈

当您需要在同一个工程目录下使用多个应用（App）管理资源栈时，可以使用 `ros-cdk --app` 命令。

1. 创建新应用。

i. 在当前目录创建 `bin/test.ts` 文件，管理资源栈 `TestStack` 。

```
#!/usr/bin/env node
import * as ros from '@alicloud/ros-cdk-core';
import { TestStack } from '../lib/test-stack';
const app = new ros.App({outdir: './cdk.out'});
new TestStack(app, 'TestStack');
app.synth();
```

ii. 在当前目录创建 `lib/test-stack.ts` 文件，在资源栈 `TestStack` 中创建ECS安全组。

```
import * as ros from '@alicloud/ros-cdk-core';
import * as ecs from '@alicloud/ros-cdk-ecs';
export class TestStack extends ros.Stack {
    constructor(scope: ros.Construct, id: string, props?: ros.StackProps) {
        super(scope, id, props);
        new ros.RosInfo(this, ros.RosInfo.description, "This is the simple ros cdk
app example.");
        // The code that defines your stack goes here
        const sg = new ecs.SecurityGroup(this, 'ros-cdk-test-sg');
    }
}
```

2. 使用新应用管理资源栈。

i. 执行以下命令，生成模板文件。

```
ros-cdk synth TestStack -j  --app "npx ts-node bin/test.ts"
```

执行命令后，输出以下内容：

```
{
  "Description": "This is the simple ros cdk app example.",
  "Metadata": {
    "ALIYUN::ROS::Interface": {
      "TemplateTags": [
        "Create by ROS CDK"
      ]
    }
  },
  "ROSTemplateFormatVersion": "2015-09-01",
  "Resources": {
    "ros-cdk-test-sg": {
      "Type": "ALIYUN::ECS::SecurityGroup"
    }
  }
}
```

ii. 执行以下命令，创建资源栈 `TestStack` 。

```
ros-cdk deploy TestStack --sync=true --app "npx ts-node bin/test.ts"
```

执行命令后，输出以下内容：

```
TestStack: deploying...
|TestStack               |2021-12-28T09:41:58 | CREATE_COMPLETE      | ALIYUN::ECS:
:SecurityGroup | sg-2ze7zn4sk5i9fgmg**** | ros-cdk-test-sg
 □ The deployment(sync deploy stack) has finished!
status: CREATE_COMPLETE
StatusReason: Stack CREATE completed successfully
StackId: 1ef89784-a728-46b2-8ec0-bc0f14c9****
```

iii. （可选）删除资源栈 `TestStack` 。

a. 执行以下命令：

```
ros-cdk destroy --sync=true --app "npx ts-node bin/test.ts"
```

b. 根据界面提示，确认删除。

```
The following stack(s) will be destroyed(Only deployed stacks will be displayed
).
TestStack
Please confirm.(Y/N)
y
```

执行命令后，输出以下内容：

```
TestStack: destroying...
|TestStack               |2021-12-28T09:41:58 | DELETE_COMPLETE      | ALIYUN::
ECS::SecurityGroup | sg-2ze7zn4sk5i9fgmg**** | ros-cdk-test-sg
 □ The task(sync destroy stack) has finished!
status: DELETE_COMPLETE
StatusReason: Stack DELETE completed successfully
StackId: 1ef89784-a728-46b2-8ec0-bc0f14c9****
```

13.6.3. 使用ROS CDK部署Nginx服务

本文以TypeScript语言为例，为您介绍如何使用ROS CDK部署Nginx服务。

步骤一：初始化工程

执行以下命令，创建工程目录并初始化工程。

```
mkdir demo
cd demo
ros-cdk init --language=typescript --generate-only=true
```

步骤二：配置阿里云凭证信息

1. 执行以下命令，配置阿里云凭证信息。

```
ros-cdk config
```

2. 根据界面提示输入配置信息。

```
ros-cdk config
endpoint(optional, default:https://ros.aliyuncs.com):
defaultRegionId(optional, default:cn-hangzhou):cn-beijing
[1] AK
[2] StsToken
[3] RamRoleArn
[4] EcsRamRole
[0] CANCEL
Authenticate mode [1...4 / 0]: 1
accessKeyId:**********
accessKeySecret:******************************
 □ Your cdk configuration has been saved successfully!
```

步骤三：安装依赖

1. 修改 `package.json` 文件，添加ECS依赖包（ros-cdk-ecs）。

```
{
  "name": "demo",
  "version": "0.1.0",
  "bin": {
    "demo": "bin/demo.js"
  },
  "scripts": {
    "build": "tsc",
    "test": "jest"
  },
  "devDependencies": {
    "@types/jest": "^25.2.1",
    "@types/node": "10.17.5",
    "typescript": "^3.9.7",
    "jest": "^25.5.0",
    "ts-jest": "^25.3.1",
    "ts-node": "^8.1.0",
    "babel-jest": "^26.6.3",
    "@babel/core": "^7.12.9",
    "@babel/preset-env": "7.12.7",
    "@babel/preset-typescript": "^7.12.7",
    "@alicloud/ros-cdk-assert": "^1.0.0"
  },
  "dependencies": {
    "@alicloud/ros-cdk-core": "^1.0.5",
    "@alicloud/ros-cdk-ecs": "^1.0.3"
  }
}
```

2. 执行以下命令，安装依赖。

```
npm install
```

步骤四：管理资源栈

1. 修改 `lib/demo-stack.ts` 文件，创建ECS实例并安装Nginx服务。

```
import * as ros from '@alicloud/ros-cdk-core';
import * as ecs from '@alicloud/ros-cdk-ecs';
import * as ROS from '@alicloud/ros-cdk-ros';
export class DemoStack extends ros.Stack {
    constructor(scope: ros.Construct, id: string, props?: ros.StackProps) {
        super(scope, id, props);
        new ros.RosInfo(this, ros.RosInfo.description, "This is the simple ros cdk app
example.");
        // The code that defines your stack goes here
        const nginxDefaultPort = new ros.RosParameter(this, 'NginxDefaultPort', {
            type: ros.RosParameterType.STRING,
        });
        const ecsInstanceType = new ros.RosParameter(this, 'EcsInstanceType', {
            type: ros.RosParameterType.STRING,
```

```
        });
        const systemDiskCategory = new ros.RosParameter(this, 'SystemDiskCategory', {
            type: ros.RosParameterType.STRING,
        });
        const vpc = new ecs.Vpc(this, 'vpc-from-ros-cdk', {
            vpcName: 'test-ros-cdk-vpc',
            cidrBlock: '10.0.0.0/8',
            description: 'This is ros cdk test'
        });
        const vsw = new ecs.VSwitch(this, `vsw-from-ros-cdk`, {
            vpcId: vpc.attrVpcId,
            // zoneId: ros.Fn.select(0, ros.Fn.getAzs(ros.Fn.ref('ALIYUN::Region'))),
            zoneId: 'cn-beijing-h',
            vSwitchName: 'test-ros-cdk-vsw',
            cidrBlock: '10.0.0.0/16',
        });
        const sg = new ecs.SecurityGroup(this, 'ros-cdk-test-sg',{
            vpcId: vpc.attrVpcId,
            securityGroupName: 'test-ros-cdk-sg',
            securityGroupIngress: [{'portRange':'22/22','nicType':'intranet','ipProtocol':'tcp','sourceCidrIp':'0.0.0.0/0'},
                {'portRange':'80/80','nicType':'intranet','ipProtocol':'tcp','sourceCidrIp':'0.0.0.0/0'}]
        });
        const ecsWaitConditionHandle = new ROS.WaitConditionHandle(this,'RosWaitConditionHandle',{
            count: 1
        })
        const ecsWaitCondition = new ROS.WaitCondition(this,'RosWaitCondition',{
            timeout: 300,
            handle: ros.Fn.ref('RosWaitConditionHandle'),
            count: 1
        })
        let NginxDefaultPort = nginxDefaultPort.valueAsString
        const ecsGroups = new ecs.InstanceGroup(this,'ros-cdk-test-ecs',{
            vpcId: vpc.attrVpcId,
            vSwitchId: vsw.attrVSwitchId,
            imageId: 'centos_7',
            maxAmount: 1,
            securityGroupId: sg.attrSecurityGroupId,
            instanceType: ecsInstanceType.valueAsString,
            systemDiskCategory: systemDiskCategory.valueAsString,
            instanceName: 'test-ros-cdk-ecs',
            userData: ros.Fn.replace({ NOTIFY: ecsWaitConditionHandle.attrCurlCli }, `#!/bin/bash\nyum install -y nginx\nsed -i \'s/listen       80;/listen       ${NginxDefaultPort};/\' /etc/nginx/nginx.conf \nsystemctl restart nginx\n NOTIFY`),
        });
    }
}
```

2. 执行以下命令，生成模板文件。

```
ros-cdk synth --json
```

执行命令后，输出以下内容：

```
{
  "Description": "This is the simple ros cdk app example.",
  "Metadata": {
    "ALIYUN::ROS::Interface": {
      "TemplateTags": [
        "Create by ROS CDK"
      ]
    }
  },
  "ROSTemplateFormatVersion": "2015-09-01",
  "Parameters": {
    "NginxDefaultPort": {
      "Type": "String"
    },
    "EcsInstanceType": {
      "Type": "String"
    },
    "SystemDiskCategory": {
      "Type": "String"
    }
  },
  "Resources": {
    "vpc-from-ros-cdk": {
      "Type": "ALIYUN::ECS::VPC",
      "Properties": {
        "CidrBlock": "10.0.0.0/8",
        "Description": "This is ros cdk test",
        "EnableIpv6": false,
        "VpcName": "test-ros-cdk-vpc"
      }
    },
    "vsw-from-ros-cdk": {
      "Type": "ALIYUN::ECS::VSwitch",
      "Properties": {
        "CidrBlock": "10.0.0.0/16",
        "VpcId": {
          "Fn::GetAtt": [
            "vpc-from-ros-cdk",
            "VpcId"
          ]
        },
        "ZoneId": "cn-beijing-h",
        "VSwitchName": "test-ros-cdk-vsw"
      }
    },
    "ros-cdk-test-sg": {
      "Type": "ALIYUN::ECS::SecurityGroup",
      "Properties": {
        "SecurityGroupIngress": [
          {
            "PortRange": "22/22",
            "SourceCidrIp": "0.0.0.0/0",
            "IpProtocol": "tcp",
            "NicType": "intranet"
```

```
        },
        {
          "PortRange": "80/80",
          "SourceCidrIp": "0.0.0.0/0",
          "IpProtocol": "tcp",
          "NicType": "intranet"
        }
      ],
      "SecurityGroupName": "test-ros-cdk-sg",
      "VpcId": {
        "Fn::GetAtt": [
          "vpc-from-ros-cdk",
          "VpcId"
        ]
      }
    }
  },
  "RosWaitConditionHandle": {
    "Type": "ALIYUN::ROS::WaitConditionHandle",
    "Properties": {
      "Count": 1,
      "Mode": "Full"
    }
  },
  "RosWaitCondition": {
    "Type": "ALIYUN::ROS::WaitCondition",
    "Properties": {
      "Handle": {
        "Ref": "RosWaitConditionHandle"
      },
      "Timeout": 300,
      "Count": 1
    }
  },
  "ros-cdk-test-ecs": {
    "Type": "ALIYUN::ECS::InstanceGroup",
    "Properties": {
      "ImageId": "centos_7",
      "InstanceType": {
        "Ref": "EcsInstanceType"
      },
      "MaxAmount": 1,
      "AllocatePublicIP": true,
      "AutoRenew": "False",
      "AutoRenewPeriod": 1,
      "InstanceChargeType": "PostPaid",
      "InstanceName": "test-ros-cdk-ecs",
      "InternetChargeType": "PayByTraffic",
      "InternetMaxBandwidthIn": 200,
      "InternetMaxBandwidthOut": 1,
      "IoOptimized": "optimized",
      "Period": 1,
      "PeriodUnit": "Month",
      "SecurityGroupId": {
```

```
          "Fn::GetAtt": [
            "ros-cdk-test-sg",
            "SecurityGroupId"
          ]
        },
        "SystemDiskCategory": {
          "Ref": "SystemDiskCategory"
        },
        "UserData": {
          "Fn::Replace": [
            {
              "NOTIFY": {
                "Fn::GetAtt": [
                  "RosWaitConditionHandle",
                  "CurlCli"
                ]
              }
            },
            {
              "Fn::Join": [
                "",
                [
                  "#!/bin/bash\nyum install -y nginx\nsed -i 's/listen       80;/listen
",
                  {
                    "Ref": "NginxDefaultPort"
                  },
                  ";/' /etc/nginx/nginx.conf \nsystemctl restart nginx\n NOTIFY"
                ]
              ]
            }
          ]
        },
        "VpcId": {
          "Fn::GetAtt": [
            "vpc-from-ros-cdk",
            "VpcId"
          ]
        },
        "VSwitchId": {
          "Fn::GetAtt": [
            "vsw-from-ros-cdk",
            "VSwitchId"
          ]
        }
      }
    }
  }
}
```

3. 执行以下命令，创建资源栈。

```
ros-cdk deploy --parameters EcsInstanceType=ecs.c6e.large --parameters NginxDefaultPort
=8080 --parameters SystemDiskCategory=cloud_essd --sync=true
```

执行命令后，输出以下内容：

```
DemoStack: deploying...
|DemoStack                |2021-12-28T09:08:38 | CREATE_COMPLETE      | ALIYUN::ECS::Sec
urityGroup | sg-2ze515oc21n45wgs**** | ros-cdk-test-sg
|DemoStack                |2021-12-28T09:08:38 | CREATE_COMPLETE      | ALIYUN::ECS::VSw
itch    | vsw-2zeuuwgla2qqrw89**** | vsw-from-ros-cdk
|DemoStack                |2021-12-28T09:08:38 | CREATE_COMPLETE      | ALIYUN::ROS::Wai
tConditionHandle |   | RosWaitConditionHandle
|DemoStack                |2021-12-28T09:08:38 | CREATE_COMPLETE      | ALIYUN::ECS::VPC
| vpc-2zedbloxtm5jetdg**** | vpc-from-ros-cdk
|DemoStack                |2021-12-28T09:08:38 | CREATE_COMPLETE      | ALIYUN::ECS::Ins
tanceGroup | i-2ze6yjdmete88**** | ros-cdk-test-ecs
|DemoStack                |2021-12-28T09:08:38 | CREATE_COMPLETE      | ALIYUN::ROS::Wai
tCondition |   | RosWaitCondition
 The deployment(sync deploy stack) has finished!
status: CREATE_COMPLETE
StatusReason: Stack CREATE completed successfully
StackId: f7ddd89b-914a-44a4-aac9-80derdg****
```

4. （可选）删除资源栈。

   i. 执行以下命令：

   ```
   ros-cdk destroy --sync=true
   ```

   ii. 根据界面提示，确认删除。

   ```
   The following stack(s) will be destroyed(Only deployed stacks will be displayed).
   DemoStack
   Please confirm.(Y/N)
   y
   ```

   执行命令后，输出以下内容：

   ```
   DemoStack: destroying...
   |DemoStack                | DELETE_COMPLETE      | ALIYUN::ECS::SecurityGroup |   | r
   os-cdk-test-sg
   |DemoStack                | DELETE_COMPLETE      | ALIYUN::ECS::VSwitch    |   | vsw-
   from-ros-cdk
   |DemoStack                | DELETE_COMPLETE      | ALIYUN::ROS::WaitConditionHandle
   |   | RosWaitConditionHandle
   |DemoStack                | DELETE_COMPLETE      | ALIYUN::ECS::VPC        |   | vpc-
   from-ros-cdk
   |DemoStack                | DELETE_COMPLETE      | ALIYUN::ECS::InstanceGroup |   | r
   os-cdk-test-ecs
   |DemoStack                | DELETE_COMPLETE      | ALIYUN::ROS::WaitCondition |   | R
   osWaitCondition
    The task(sync destroy stack) has finished!
   status: DELETE_COMPLETE
   StatusReason: Stack DELETE completed successfully
   StackId: f7ddd89b-914a-44a4-aac9-80derdg****
   ```

13.6.4. 使用ROS CDK批量创建多个资源

本文以TypeScript语言为例，为您介绍如何使用ROS CDK批量创建多个资源。

步骤一：初始化工程

执行以下命令，创建工程目录并初始化工程。

```
mkdir demo
cd demo
ros-cdk init --language=typescript --generate-only=true
```

步骤二：配置阿里云凭证信息

1. 执行以下命令，配置阿里云凭证信息。

   ```
   ros-cdk config
   ```

2. 根据界面提示输入配置信息。

   ```
   ros-cdk config
   endpoint(optional, default:https://ros.aliyuncs.com):
   defaultRegionId(optional, default:cn-hangzhou):cn-beijing
   [1] AK
   [2] StsToken
   [3] RamRoleArn
   [4] EcsRamRole
   [0] CANCEL
   Authenticate mode [1...4 / 0]: 1
   accessKeyId:************************
   accessKeySecret:******************************
    □ Your cdk configuration has been saved successfully!
   ```

步骤三：安装依赖

1. 修改 `package.json` 文件，添加ECS依赖包（ros-cdk-ecs）。

```
{
  "name": "demo",
  "version": "0.1.0",
  "bin": {
    "demo": "bin/demo.js"
  },
  "scripts": {
    "build": "tsc",
    "test": "jest"
  },
  "devDependencies": {
    "@types/jest": "^25.2.1",
    "@types/node": "10.17.5",
    "typescript": "^3.9.7",
    "jest": "^25.5.0",
    "ts-jest": "^25.3.1",
    "ts-node": "^8.1.0",
    "babel-jest": "^26.6.3",
    "@babel/core": "^7.12.9",
    "@babel/preset-env": "7.12.7",
    "@babel/preset-typescript": "^7.12.7",
    "@alicloud/ros-cdk-assert": "^1.0.0"
  },
  "dependencies": {
    "@alicloud/ros-cdk-core": "^1.0.5",
    "@alicloud/ros-cdk-ecs": "^1.0.3"
  }
}
```

2. 执行以下命令，安装依赖。

```
npm install
```

步骤四：管理资源栈

1. 修改 `lib/demo-stack.ts` 文件，创建安全组。

   以下代码通过for循环创建安全组入方向的访问规则（SecurityGroupIngress），实现了批量创建多个资源的诉求，并且代码更为精简。

```
import * as ros from '@alicloud/ros-cdk-core';
import * as ecs from '@alicloud/ros-cdk-ecs';
export class CdkDemoStack extends ros.Stack {
  constructor(scope: ros.Construct, id: string, props?: ros.StackProps) {
    super(scope, id, props);
    new ros.RosInfo(this, ros.RosInfo.description, "This is the simple ros cdk app exam
ple.");
    // The code that defines your stack goes here
    const sg = new ecs.SecurityGroup(this, 'ros-cdk-test-sg');
    let PortList= ['22','3389','80'];
    for (let port of PortList) {
       new ecs.SecurityGroupIngress(this, `ros-cdk-test-sg-ingress-${port}`, {
        portRange: `${port}/${port}`,
        nicType: 'intranet',
        sourceCidrIp: '0.0.0.0/0',
        ipProtocol: 'tcp',
        securityGroupId: sg.attrSecurityGroupId
      },true);
    }
  }
}
```

2. 执行以下命令，生成模板文件。

```
ros-cdk synth --json
```

执行命令后，输出以下内容：

```
{
  "Description": "This is the simple ros cdk app example.",
  "Metadata": {
    "ALIYUN::ROS::Interface": {
      "TemplateTags": [
        "Create by ROS CDK"
      ]
    }
  },
  "ROSTemplateFormatVersion": "2015-09-01",
  "Resources": {
    "ros-cdk-test-sg": {
      "Type": "ALIYUN::ECS::SecurityGroup"
    },
    "ros-cdk-test-sg-ingress-22": {
      "Type": "ALIYUN::ECS::SecurityGroupIngress",
      "Properties": {
        "IpProtocol": "tcp",
        "PortRange": "22/22",
        "NicType": "intranet",
        "Priority": 1,
        "SecurityGroupId": {
          "Fn::GetAtt": [
            "ros-cdk-test-sg",
            "SecurityGroupId"
          ]
        },
```

```
        "SourceCidrIp": "0.0.0.0/0"
      }
    },
    "ros-cdk-test-sg-ingress-3389": {
      "Type": "ALIYUN::ECS::SecurityGroupIngress",
      "Properties": {
        "IpProtocol": "tcp",
        "PortRange": "3389/3389",
        "NicType": "intranet",
        "Priority": 1,
        "SecurityGroupId": {
          "Fn::GetAtt": [
            "ros-cdk-test-sg",
            "SecurityGroupId"
          ]
        },
        "SourceCidrIp": "0.0.0.0/0"
      }
    },
    "ros-cdk-test-sg-ingress-80": {
      "Type": "ALIYUN::ECS::SecurityGroupIngress",
      "Properties": {
        "IpProtocol": "tcp",
        "PortRange": "80/80",
        "NicType": "intranet",
        "Priority": 1,
        "SecurityGroupId": {
          "Fn::GetAtt": [
            "ros-cdk-test-sg",
            "SecurityGroupId"
          ]
        },
        "SourceCidrIp": "0.0.0.0/0"
      }
    }
  }
}
```

3. 执行以下命令，创建资源栈。

```
ros-cdk deploy --sync=true
```

执行命令后，输出以下内容：

```
DemoStack: deploying...
|DemoStack               |2021-12-28T09:17:54 | CREATE_COMPLETE      | ALIYUN::ECS::Sec
urityGroupIngress | sg-2ze7gbyb0sngx7et**** | ros-cdk-test-sg-ingress-22
|DemoStack               |2021-12-28T09:17:54 | CREATE_COMPLETE      | ALIYUN::ECS::Sec
urityGroupIngress | sg-2ze7gbyb0sngx73a**** | ros-cdk-test-sg-ingress-80
|DemoStack               |2021-12-28T09:17:54 | CREATE_COMPLETE      | ALIYUN::ECS::Sec
urityGroup | sg-2ze7gbyb0sngx7r4**** | ros-cdk-test-sg
|DemoStack               |2021-12-28T09:17:54 | CREATE_COMPLETE      | ALIYUN::ECS::Sec
urityGroupIngress | sg-2ze7gbyb0snget3**** | ros-cdk-test-sg-ingress-3389
 □ The deployment(sync deploy stack) has finished!
status: CREATE_COMPLETE
StatusReason: Stack CREATE completed successfully
StackId: a6f1fd14-c985-4dd2-9913-1e7bbeef****
```

4. （可选）删除资源栈。

   i. 执行以下命令：

   ```
   ros-cdk destroy --sync=true
   ```

   ii. 根据界面提示，确认删除。

   ```
   The following stack(s) will be destroyed(Only deployed stacks will be displayed).
   DemoStack
   Please confirm.(Y/N)
   y
   ```

   执行命令后，输出以下内容：

   ```
   DemoStack: destroying...
   |DemoStack               | DELETE_COMPLETE      | ALIYUN::ECS::SecurityGroupIngress
   |  | ros-cdk-test-sg-ingress-22
   |DemoStack               | DELETE_COMPLETE      | ALIYUN::ECS::SecurityGroupIngress
   |  | ros-cdk-test-sg-ingress-80
   |DemoStack               | DELETE_COMPLETE      | ALIYUN::ECS::SecurityGroup |  | r
   os-cdk-test-sg
   |DemoStack               | DELETE_COMPLETE      | ALIYUN::ECS::SecurityGroupIngress
   |  | ros-cdk-test-sg-ingress-3389
   □ The task(sync destroy stack) has finished!
   status: DELETE_COMPLETE
   StatusReason: Stack DELETE completed successfully
   StackId: a6f1fd14-c985-4dd2-9913-1e7bbeef****
   ```

14.视频专区