aws-service-catalog-puppet

Eamonn Faherty

Aug 25, 2023

CONTENTS

CHAPTER

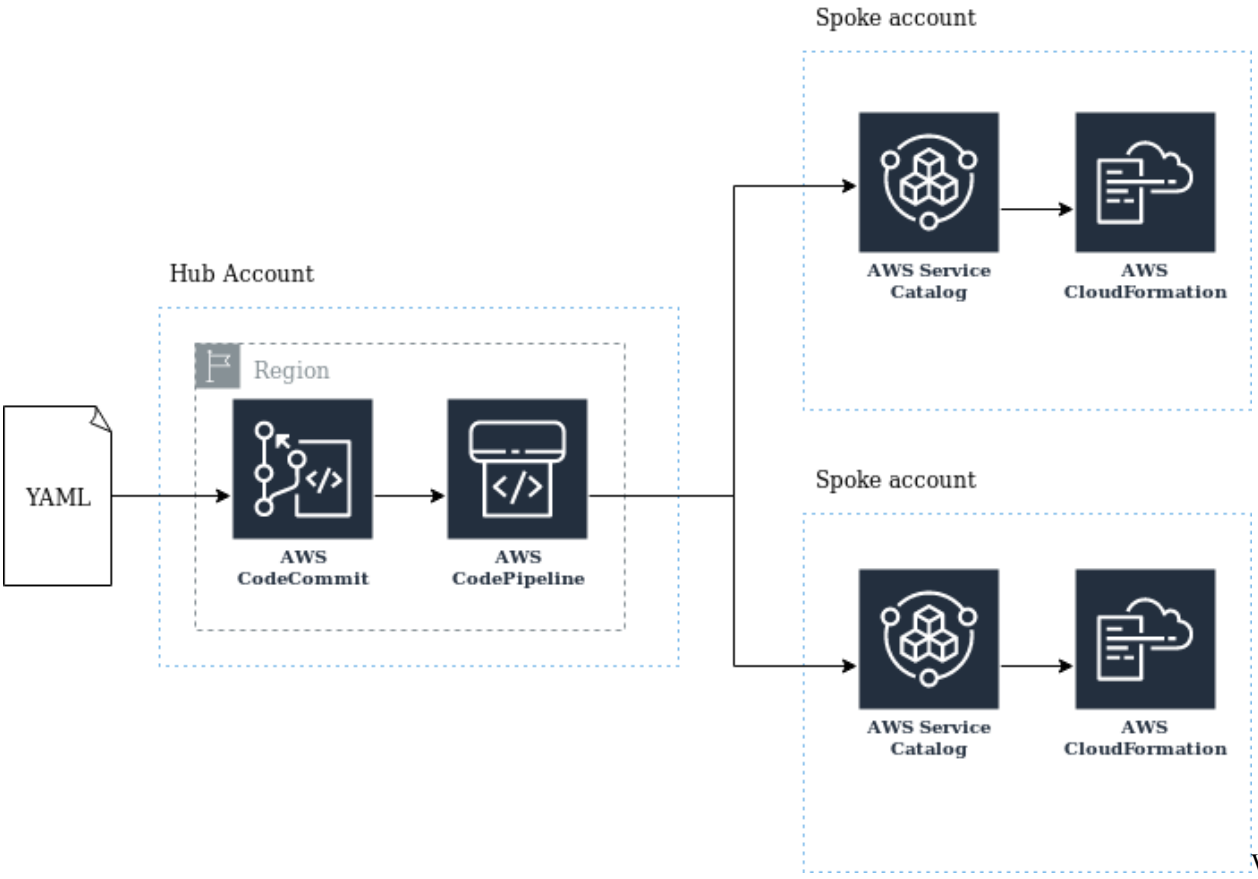
ONE

WHAT IS THIS?

Service Catalog Puppet is a framework that enables you to provision service catalog products into accounts that need to have them. You declare your service catalog products and the accounts you want them to be available in via a configuration file. Service Catalog Puppet then walks through this configuration file and determines which products need to be made available in which accounts. You can use tags or account numbers to indicate which products should be available in which accounts. For example if using tags for both accounts and products, products tagged dev will be made available in accounts tagged dev.

The framework works through your lists, dedupes and spots collisions and then provisions the products into your AWS accounts for you. It handles Service Portfolio sharing, accepting Portfolio shares and can provision products cross account and cross region.

1.1 High level architecture diagram



What is this

You use an AWS CodeBuild project in a central *puppet* account that provisions AWS Service Catalog Products into *spoke* accounts on your behalf. The framework takes care of cross account sharing and cross region product replication for you.

CHAPTER

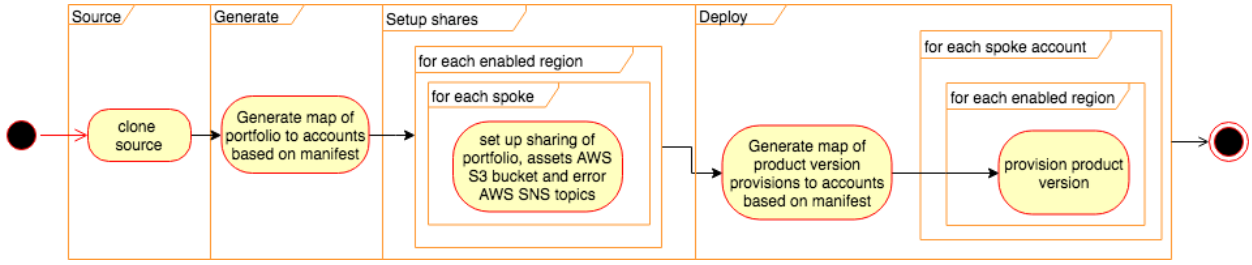
TWO

GETTING UP AND RUNNING

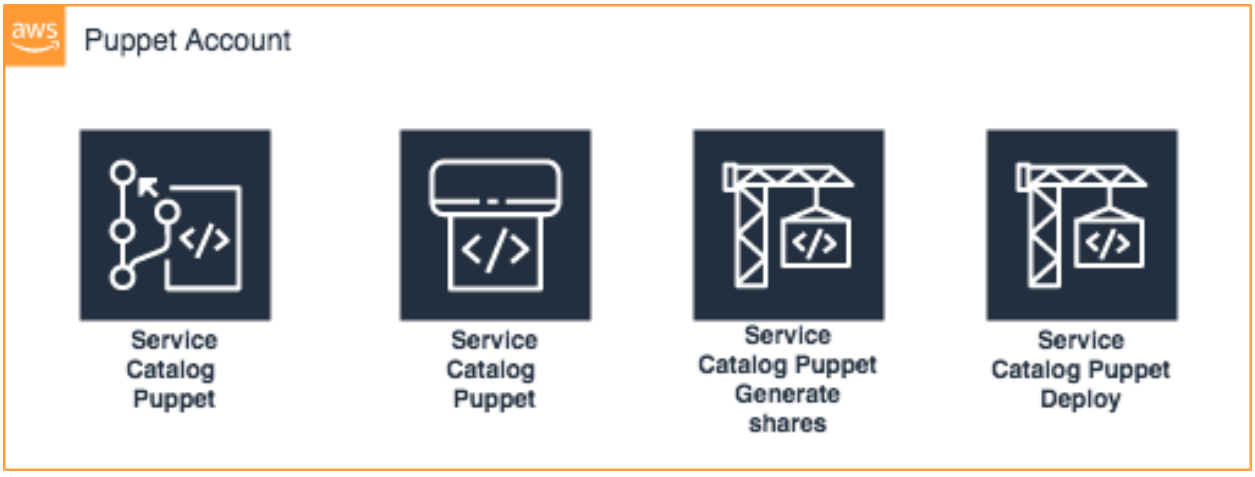
ServiceCatalog-Puppet runs in your AWS Account. In order for you to install it into your account you can use the aws-service-catalog-puppet cli. This is distributed via [PyPi](https://pypi.org/project/aws-service-catalog-puppet/)

2.1 What am I going to install?

Once you have completed the install you will have the following pipeline in your account:



using the following services:



2.2 Before you install

You should consider which account will be the home for your puppet. This account will contain the AWS CodePipelines and will need to be accessible to any accounts you would like to share with. If you are using ServiceCatalog-Factory, we recommend you install both tools into the same account.

2.3 Installing

see https://service-catalog-tools-workshop.com/installation.html

CHAPTER

THREE

DESIGNING YOUR MANIFEST

3.1 Purpose of the manifest file

The manifest file is there to describe what you want to provision and into which accounts you want to provision products into. It is possible to use AWS Organizations to make your manifest file more concise and easier to work with but the premise is the same - it is just a list of accounts and AWS Service Catalog products.

3.2 Sections of the manifest file

There are three sections to a manifest file - the global parameters, the accounts list and the launches. Each of the three are described in the following sections.

3.2.1 Parameters

It is possible to specify global parameters that should be used when provisioning your AWS Service Catalog Products. You can set the value to an explicit value or you can set the value to the result of a function call - using funcation calls to set parameter values is known as using a macro.

Here is an example of a simple global parameter:

```
schema: puppet-2019-04-01

parameters:
    CloudTrailLoggingBucketName:
      default: cloudtrail-logs-for-aws
```

It is possible to also specify a parameter at the account level:

```
accounts:
  - account_id: '<YOUR_ACCOUNT_ID>'
    name: '<YOUR_ACCOUNT_NAME>'
    default_region: eu-west-1
    regions_enabled:
      - eu-west-1
      - eu-west-1
    tags:
      - type:prod
      - partition:eu
      - scope:pci
    parameters:
```

(continues on next page)

(continued from previous page)

```
      RoleName:
        default: DevAdmin
      Path:
        default: /human-roles/
```

And finally you specify parameters at the launch level:

```
launches:
  account-iam-for-prod:
    portfolio: demo-central-it-team-portfolio
    product: account-iam
    version: v1
    parameters:
      RoleName:
        default: DevAdmin
      Path:
        default: /human-roles/
    deploy_to:
      tags:
        - tag: type:prod
          regions: default_region
```

Whenever Puppet provisions a product it checks the parameters for the product. If it sees the name match one of the parameter values it will use it. In order to avoid clashes with parameter names we recommend using descriptive names like in the example - using the parameter names like `BucketName` will lead you into trouble pretty quickly.

The order of precedence for parameters is account level parameters override all others and launch level parameters override global.

Retrieving AWS SSM Parameters

Note: This was added in version 0.0.33

You can retrieve parameter values from SSM in the puppet account - these parameters do not belong to the spoke account. Here is an an example:

```
schema: puppet-2019-04-01

parameters:
    CentralLoggingBucketName:
      ssm:
        name: central-logging-bucket-name
```

You can get a different value for each region:

```
schema: puppet-2019-04-01

parameters:
    CentralLoggingBucketName:
      ssm:
        name: central-logging-bucket-name
        region: eu-west-1
```

Note: Since 0.94.0 you can use intrinsic functions within the parameter name

You can use intrinsic functions for AWS AccountId and Region within the ssm parameter name:

```
schema: puppet-2019-04-01

parameters:
    VPCCidrRange:
      ssm:
        name: /vpcs/${AWS::AccountId}/${AWS::Region}/cidr
```

${AWS::AccountId} and ${AWS::Region} will be replaced with the spoke account id and the region where the provisioning will occur. This allows you to build out account specific parameters using SSM parameter store parameters in the hub account.

You can use your account vending machine to provision the account specific ssm parameters in the hub account and then use this to read them.

Setting AWS SSM Parameters

Note: This was added in version 0.0.34

You can set the value of an SSM Parameter to the output of a CloudFormation stack output:

```
account-iam-sysops:
  portfolio: demo-central-it-team-portfolio
  product: account-iam
  version: v1
  parameters:
    Path:
      default: /human-roles/
    RoleName:
      default: SysOps
  deploy_to:
    tags:
    - regions: default_region
      tag: type:prod
  outputs:
    ssm:
      -  param_name: account-iam-sysops-role-arn
         stack_output: RoleArn
```

The example above will provision the product `account-iam` into an account. Once the stack has been completed it will get the value of the output named `RoleArn` of the CloudFormation stack and insert it into SSM within the default region using a parameter name of `account-iam-sysops-role-arn`

You can also set override which region the output is read from and which region the SSM parameter is written to:

```
account-iam-sysops:
  portfolio: demo-central-it-team-portfolio
  product: account-iam
  version: v1
  parameters:
```

(continues on next page)

(continued from previous page)

```
    Path:
      default: /human-roles/
    RoleName:
      default: SysOps
  deploy_to:
    tags:
    - regions: default_region
      tag: type:prod
  outputs:
    ssm:
      -  param_name: account-iam-sysops-role-arn
         stack_output: RoleArn
         region: us-east-1
```

Note: There is currently no capability of reading a value from a CloudFormation stack from one region and setting an SSM param in another.

Macros

You can also use a macro to set the value of a parameter. It works in the same way as a normal parameter except it executes a function to get the value first. Here is an an example:

```
schema: puppet-2019-04-01

parameters:
    AllAccountIds:
      macro:
        method: get_accounts_for_path
        args: /
```

At the moment there are the following macros supported:

```
+------------------------+------------------------------+------------------------------
↪-----------------+
| macro method name      | args                         | description                  ␣
↪                  |
+========================+==============================+==========================================
| get_accounts_for_path  | ou path to get accounts for  | Returns a comma seperated␣
↪list of account ids|
+------------------------+------------------------------+------------------------------
↪-----------------+
```

3.2.2 Mappings

Note: This was added in version 0.92.0

Within the mappings section you can define mappings that can be used to set the value of parameters.

Here a mapping is defined:

```yaml
schema: puppet-2019-04-01

mappings:
  WebProxyDevice:
    us-east-1:
      "ami": "ami-15f77f867"
    us-west-1:
      "ami": "ami-0bdb82235"
    eu-west-1:
      "ami": "ami-16506cd98"
```

Here a mapping is used:

```yaml
schema: puppet-2019-04-01

launches:
  DeployProxy:
    deploy_to:
      tags:
      - regions: enabled_regions
        tag: ou:prod
    parameters:
      AMI:
        mapping:
        - WebProxyDevice
        - AWS::Region
        - ami
    portfolio: networking
    product: proxy
    version: v3
```

When DeployProxy is provisioned the parameter named AMI will have its value determined. It's value will be taken from the mappings section. The framework will look through the mappings for one named WebProxyDevice. The framework will then look within the WebProxyDevice dictionary for an object with the name of the current region and then within that for an item with the key ami.

It is also possible to use AWS::AccountId:

```yaml
schema: puppet-2019-04-01

mappings:
  AccountDetails:
    0123456789010:
      "owner": "appteam1@example.com"
```

You can also combine them:

```yaml
schema: puppet-2019-04-01

mappings:
  Networking:
    0123456789010:
      eu-west-1:
        cidr: "10.0.0.0/24"
```

You can use the special value of default as a catch all when you do you not specify a value:

```
schema: puppet-2019-04-01

mappings:
  AccountDetails:
    owners:
        0123456789010: "appteam1@example.com"
        0098765432102: "appteam2@example.com"
        default: "cloudteam@example.com"
```

3.2.3 Accounts

With the accounts section, you can describe your AWS accounts. You can set a default region, the enabled regions and you can tag your accounts. This metadata describing your account is used to determine which packages get deployed into your accounts.

Setting a default region

Within your account you may have a _home_ or a default region. This may be the closest region to the team using the account. You use `default_region` when describing your account and then you can use `default_region` again as a target when you specify your product launches - the product will be provisioned into the region specified.

Here is an example with a `default_region` set to `us-east-1`:

```
schema: puppet-2019-04-01

accounts:
  - account_id: '<YOUR_ACCOUNT_ID>'
    name: '<YOUR_ACCOUNT_NAME>'
    default_region: us-east-1
    regions_enabled:
      - us-east-1
      - us-west-2
    tags:
      - type:prod
      - partition:us
      - scope:pci
```

Note: Please note `default_region` can only be a string - not a list.

Setting enabled regions

You may chose not to use every region within your AWS Account. When describing an AWS account you can specify which regions are enabled for an account using `regions_enabled`.

Here is an example with `regions_enabled` set to `us-east-1 and us-west-2`:

```
schema: puppet-2019-04-01

accounts:
  - account_id: '<YOUR_ACCOUNT_ID>'
    name: '<YOUR_ACCOUNT_NAME>'
```

(continues on next page)

(continued from previous page)

```
    default_region: us-east-1
    regions_enabled:
      - us-east-1
      - us-west-2
    tags:
      - type:prod
      - partition:us
      - scope:pci
```

Note: Please note `regions_enabled` can only be a list of strings - not a single string

Setting tags

You can describe your account using tags. Tags are specified using a list of strings. We recommend using namespaces for your tags, adding an extra dimension to them. If you choose to do this you can use a colon to split name and values.

Here is an example with namespaced tags:

```
schema: puppet-2019-04-01

accounts:
  - account_id: '<YOUR_ACCOUNT_ID>'
    name: '<YOUR_ACCOUNT_NAME>'
    default_region: us-east-1
    regions_enabled:
      - us-east-1
      - us-west-2
    tags:
      - type:prod
      - partition:us
      - scope:pci
```

In this example there the following tags: - namespace of type and value of prod - namespace of partition and value of us - namespace of scope and value of pci.

The goal of tags is to provide a classification for your accounts that can be used to a deployment time.

Using an OU id or path (integration with AWS Organizations)

Note: This was added in version 0.0.18

When specifying an account you can use short hand notation of `ou` instead of `account_id` to build out a list of accounts with the same properties.

For example you can use an AWS Organizations path:

```
schema: puppet-2019-04-01

accounts:
  - ou: /prod
    name: '<CHOOSE A NAME FOR YOUR ACCOUNTS LIST>'
```

(continues on next page)

(continued from previous page)

```
    default_region: us-east-1
    regions_enabled:
      - us-east-1
      - us-west-2
    tags:
      - type:prod
      - partition:us
      - scope:pci
```

The framework will get a list of all AWS accounts within the `/prod` Organizational unit and expand your manifest to look like the following (assuming accounts 0123456789010 and 0109876543210 are the only accountss within `/prod`):

```
schema: puppet-2019-04-01

accounts:
  - account_id: 0123456789010
    name: '<YOUR_ACCOUNT_NAME>'
    default_region: us-east-1
    regions_enabled:
      - us-east-1
      - us-west-2
    tags:
      - type:prod
      - partition:us
      - scope:pci
  - account_id: 0109876543210
    name: '<YOUR_ACCOUNT_NAME>'
    default_region: us-east-1
    regions_enabled:
      - us-east-1
      - us-west-2
    tags:
      - type:prod
      - partition:us
      - scope:pci
```

3.2.4 Launches

Launches allow you to decide which products get provisioned into each account. You link product launches to accounts using tags or explicit account ids and you can set which regions the products are launched into.

Timeouts

Note: This was added in version 0.1.14

If you are worried that a launch may fail and take a long time to fail you can set a timeout `timeoutInSeconds`:

```
schema: puppet-2019-04-01

launches:
  account-iam-for-prod:
```

(continues on next page)

(continued from previous page)

```
    portfolio: example-simple-central-it-team-portfolio
    product: account-iam
    timeoutInSeconds: 10
    version: v1
    deploy_to:
      tags:
        - tag: type:prod
          regions: default_region
```

Tag based launches

You can specify a launch to occur using `tags` in the `deploy_to` section of a launch.

Here is an example, it deploys a `v1` of a product named `account-iam` from the portfolio `example-simple-central-it-team-portfolio` into into the `default_region` of all accounts tagged `type:prod`:

```
schema: puppet-2019-04-01

launches:
  account-iam-for-prod:
    portfolio: example-simple-central-it-team-portfolio
    product: account-iam
    version: v1
    deploy_to:
      tags:
        - tag: type:prod
          regions: default_region
```

When you specify more than one tag entry in deploy_to->tags the framework will interpret this as an or so the following snippet will provision `v1` of `account-iam` to all accounts tagged `type:prod` or `type:dev`:

```
schema: puppet-2019-04-01

launches:
  account-iam-for-prod:
    portfolio: example-simple-central-it-team-portfolio
    product: account-iam
    version: v1
    deploy_to:
      tags:
        - tag: type:prod
          regions: default_region
        - tag: type:dev
          regions: default_region
```

Account based launches

You can also specify a launch to occur explicity in an account by using the `accounts` section in the `deploy_to` section of a launch.

Here is an example, it deploys a `v1` of a product named `account-iam` from the portfolio `example-simple-central-it-team-portfolio` into into the `default_region` of the accounts `0123456789010`:

```yaml
schema: puppet-2019-04-01

launches:
  account-iam-for-prod:
    portfolio: example-simple-central-it-team-portfolio
    product: account-iam
    version: v1
    deploy_to:
      accounts:
        - account_id: '0123456789010'
          regions: default_region
```

Choosing which regions to provision into

When writing your launches you can choose which regions you provision into.

The valid values for regions are:

- `enabled` - this will deploy to each enabled region for the account
- `regions_enabled` - this will deploy to each enabled region for the account
- `default_region` - this will deploy to the default region specified for the account
- `all` - this will deploy to all regions enabled in your config (whilst setting up Puppet)
- list of AWS regions - you can type in a list of AWS regions (each region selected should be present in your config)

Dependencies between launches

Where possible we recommend building launches to be independent. However, there are cases where you may need to setup a hub account before setting up a spoke or there may be times you are using AWS Lambda to back AWS CloudFormation custom resources. In these examples it would be beneficial to be able to say deploy launch x and then launch y. To achieve this You can use `depends_on` within your launch like so:

```yaml
launches:
  account-vending-account-creation:
    portfolio: demo-central-it-team-portfolio
    product: account-vending-account-creation
    version: v1
    depends_on:
      - account-vending-account-bootstrap-shared
      - account-vending-account-creation-shared
    deploy_to:
      tags:
        - tag: scope:puppet-hub
          regions: default_region

  account-vending-account-bootstrap-shared:
    portfolio: demo-central-it-team-portfolio
    product: account-vending-account-bootstrap-shared
    version: v1
    deploy_to:
      tags:
        - tag: scope:puppet-hub
```

(continues on next page)

(continued from previous page)

```
          regions: default_region

  account-vending-account-creation-shared:
    portfolio: demo-central-it-team-portfolio
    product: account-vending-account-creation-shared
    version: v1
    deploy_to:
      tags:
        - tag: scope:puppet-hub
          regions: default_region
```

In this example the framework will deploy `account-vending-account-creation` only when `account-vending-account-bootstrap-shared` and `account-vending-account-creation-shared` have been attempted.

Termination of products

Note: This was added in version 0.1.11

To terminate the provisioned product from a spoke account (which will delete the resources deployed) you can change the status of the launch using the `status` keyword:

```
launches:
  account-vending-account-creation:
    portfolio: demo-central-it-team-portfolio
    product: account-vending-account-creation
    version: v1
    status: terminated
    deploy_to:
      tags:
        - tag: scope:puppet-hub
          regions: default_region
```

When you mark a launch as terminated and run your pipeline the resources will be deleted and you can then remove the launch from your manifest. Leaving it in will not cause any errors but will result in your pipeline running time to be longer than it needs to be.

Please note, when mark your launch as `terminated` it cannot have dependencies, parameters or outputs. Leaving these in will cause the termination action to fail.

Note: When you set status to terminated you must remove your depends_on and parameters for it to work.

Warning: Since 0.1.16, terminating a product will also remove any SSM Parameters you created for it via the manifest.yaml

Managing large manifests or working in teams (multiple manifest files)

Note: This was added in version 0.71.0

If you have a large manifest file or are working in a team you may find it difficult managing changes occurring to your manifest file. You may find yourself having a lot of merge conflicts. To resolve this you can split your manifest file into smaller pieces. You can specify launches in a launch directory within your ServiceCatalogPuppet repository:

```
 tree ServiceCatalogPuppet
ServiceCatalogPuppet
├── launches
│   └── launches-for-team-a.yaml
├── manifest.yaml
```

The file (in this example launches-for-team-a.yaml) should be a list of launches:

```
 cat launches-for-team-a.yaml
account-vending-account-creation:
    portfolio: demo-central-it-team-portfolio
    product: account-vending-account-creation
    version: v1
    depends_on:
      - account-vending-account-bootstrap-shared
      - account-vending-account-creation-shared
    deploy_to:
      tags:
        - tag: scope:puppet-hub
          regions: default_region

account-vending-account-bootstrap-shared:
    portfolio: demo-central-it-team-portfolio
    product: account-vending-account-bootstrap-shared
    version: v1
    deploy_to:
      tags:
        - tag: scope:puppet-hub
          regions: default_region
```

The framework will load the manifest.yaml and *overwrite* any launches with ones defined in files from the launches directory. The framework will not warn you of any overrides.

You can also specify parameters and spoke-local-portfolios in directories too. When doing so, the files should contain lists of parameters or spoke-local-portfolios and should not be a dictionary.

```
 tree ServiceCatalogPuppet
ServiceCatalogPuppet
├── parameters
│   └── parameters-for-team-a.yaml
├── spoke-local-portfolios
│   └── spoke-local-portfolios-for-team-a.yaml
├── manifest.yaml
```

The names of the file within the launches, parameters and spoke-local-portfolios are ignored.

You can also declare other manifest files in a manifests directory:

```
 tree ServiceCatalogPuppet
ServiceCatalogPuppet
```

(continues on next page)

(continued from previous page)

```
├── manifests
    └── manifest-for-team-a.yaml
    └── manifest-for-networking.yaml
    └── manifest-for-governance.yaml
```

When you write a manifest file in the manifests directory the accounts section is ignored - you can only specify launches, parameters and spoke-local-portfolios.

Managing large manifests or working across multiple environments (external versions / properties files)

Note: This was added in version 0.76.0

If you are using puppet to manage multiple environments you may find it easier to keep the versions of your launches in properties files instead of the manifest.yaml files. To do this you create a file named manifest.properties in the same directory as your manifest.yaml file. Within this file you can specify the following:

```
[launches]
IAM-1.version = v50
```

This will set the version for the launch with the name IAM-1 to v50.

Please note this will overwrite the values specified in the manifest.yaml files with no warning.

If you are using multiple instances of puppet you can also create a file named manifest-<puppet-account-id>.properties. Values in this file will overwrite all other values making the order of reading:

1. manifest.yaml
2. files in manifests/
3. manifest.properties
4. manifest-<puppet-account-id>.properties

Sharing mode

Note: This was added in version 0.88.0

When you write a launch, the framework will share the portfolio used with each spoke account you are deploying into. The framework shares with each account and accepts the share within each account. You can tell the framework to share with an OU (using Organizational sharing) instead and then accept the share from within each account still. This reduces the time taken to share portfolios but means all accounts in the same OU will have the portfolio shared with them - those account will not have the portfolio share accepted. To enable this behaviour you need to set the sharing_mode:

```
launches:
  account-iam-for-prod:
    portfolio: example-simple-central-it-team-portfolio
    product: account-iam
    version: v1
    sharing_mode: AWS_ORGANIZATIONS
```

(continues on next page)

(continued from previous page)

```
    deploy_to:
      tags:
        - tag: type:prod
          regions: default_region
```

To revert back you can set sharing_mode back to ACCOUNT:

```
launches:
  account-iam-for-prod:
    portfolio: example-simple-central-it-team-portfolio
    product: account-iam
    version: v1
    sharing_mode: ACCOUNT
    deploy_to:
      tags:
        - tag: type:prod
          regions: default_region
```

If you are using this feature you must be able to share using Organizations in your puppet account. To do this you must have installed puppet into your AWS Organizations management account or you must have delegated your puppet account as an AWS Service Catalog organizations master account.

The default value for sharing_mode is ACCOUNT unless you change it using the following command

```
servicecatalog-puppet set-config-value global_sharing_mode_default AWS_ORGANIZATIONS
```

Alternatively, you can also add the following to your config:

```
global_sharing_mode_default: AWS_ORGANIZATIONS
```

When you change the global_sharing_mode_default it affects launches and spoke-local-portfolios.

3.2.5 Lambda Invocations

Note: This was added in version 0.83.0

If you are migrating to puppet from your own AWS Lambda and AWS Step Functions solution you may want to reuse some of your Lambda functions to orchestrate activities like the removal of default VPCs or other actions in your accounts where using AWS Service Catalog + AWS CloudFormation may be cumbersome. To do this you can use `lambda-invocation` in your manifest file:

```
lambda-invocations:
  remove-default-vpc:
    function_name: remove-default-vpc
    qualifier: $LATEST
    invocation_type: Event
    invoke_for:
      tags:
        - regions: enabled_regions
          tag: scope:all
```

The above example will build a list by walking through each `enabled_region` for all accounts tagged `scope:all`. It will then invoke the `$LATEST` version of the `remove-default-vpc` in your puppet account

for each item in the list, setting the parameters in the event object of the designated lambda to include `account_id` and `region` properties so you can implement whatever you want.

```
lambda-invocations:
  remove-default-vpc:
    function_name: remove-default-vpc
    qualifier: $LATEST
    invocation_type: Event
    depends_on:
      - name: remove-default-vpc-lambda
        type: launch
    invoke_for:
      tags:
        - regions: enabled_regions
          tag: scope:all
```

The `lambda-invocations` section includes support for depends_on where you can depend on another `lambda-invocations` or a `launch`. Using the depends_on you can provision the AWS Lambda function before executing using puppet as your complete solution for configuration.

The properties for `function_name`, `qualifier` and `invocation_type` are passed as is to the AWS Boto3 Lambda invoke function.

You can use parameters as you can for launches:

```
lambda-invocations:
  remove-default-vpc:
    function_name: remove-default-vpc
    qualifier: $LATEST
    invocation_type: Event
    parameters:
      RoleName:
        default: DevAdmin
      CentralLoggingBucketName:
        ssm:
          name: central-logging-bucket-name
          region: eu-west-1
    depends_on:
      - name: remove-default-vpc-lambda
        type: launch
    invoke_for:
      tags:
        - regions: enabled_regions
          tag: scope:all
```

If you set the invocation_type to Event puppet will not check if the Lambda function completed successfully. If you set the invocation_type to RequestResponse then it will wait for completion and error should the function not exit successfully.

CHAPTER

FOUR

BOOTSTRAPPING SPOKES

ServiceCatalog-Puppet runs in a hub and spoke model. Your spokes must be bootstrapped so that they can work with the puppet account.

You have some options when it comes to bootstrapping spokes.

4.1 Individual Spokes

To bootstrap a spoke you can use the cli

4.1.1 Bootstrap your spoke

You will need to bootstrap each of your spokes. In order to do so please export your credentials and set your profile, then run the following:

```
servicecatalog-puppet bootstrap-spoke <ACCOUNT_ID_OF_YOUR_PUPPET>
```

4.1.2 Bootstrap your spoke as

If your current credentials are not for a role in the account you want to bootstrap but you can assume a role to get there then you can use bootstrap-spoke-as:

```
servicecatalog-puppet bootstrap-spoke-as <ACCOUNT_ID_OF_YOUR_PUPPET> <ARN_OF_IAM_ROLE_
↪TO_ASSUME_BEFORE_BOOTSTRAPPING>
```

You can specify multiple ARNs and they will be assumed from left to right before bootstrapping:

```
servicecatalog-puppet bootstrap-spoke-as <ACCOUNT_ID_OF_YOUR_PUPPET> <ARN_1> <ARN_2>
↪<ARN_3> <ARN_4>
```

With the above example the framework will assume the role defined in ARN_1, then assume the role in ARN_2, then assume the role in ARN_3 and finally then assume the role in ARN_4 before bootstrapping.

4.2 Groups of Spokes

You can bootstrap groups of accounts also.

4.2.1 Bootstrap all accounts in an OU via the cli

You should export credentials that allow you to list accounts in your org and allow you to assume a role in the spoke accounts so they can be bootstrapped. Once you have done this you can use:

```
servicecatalog-puppet bootstrap-spokes-in-ou <OU_PATH_OR_ID> <NAME_OF_IAM_ROLE_TO_
↪ASSUME_BEFORE_BOOTSTRAPPING>
```

The name of IAM role you specify should have the permission in the target ou accounts to bootstrap the spokes. For example, the following will assume the role DevOpsAdminRole in the spokes contained in the /dev ou in order to bootstrap:

```
servicecatalog-puppet bootstrap-spokes-in-ou /dev DevOpsAdminRole
```

If your current role does not allow you to list accounts in the org or allow you to assume role cross account you can specify an ARN of a role that does. When you do so the framework will assume that role first and then perform the bootstrapping.

```
servicecatalog-puppet bootstrap-spokes-in-ou /dev DevOpsAdminRole␣
↪arn:aws:iam::0123456789010:role/OrgUserRole
```

Note: bootstrap-spokes-in-ou was added in version 0.44.0

4.2.2 Bootstrap all accounts in an OU via CodeBuild

In your account you can find an AWS CodeBuild project named: *servicecatalog-puppet-bootstrap-an-ou*

You start a run of that project and override the environmental variables to use bootstrap-spokes-in-ou.

Note: bootstrap-spokes-in-ou was added in version 0.44.0

4.3 Restricting Spokes

When you are bootstrapping a spoke you can specify the ARN of an IAM Permission Boundary that should be applied to the spokes PuppetRole. This permission boundary should permit the PuppetRole to interact with AWS Service Catalog to accept shares, manage portfolios and to add, provision and terminate products. In addition the role should allow the use of AWS SNS, AWS EventBridge, AWS OpsCenter.

To use the boundary you must bootstrap a spoke using an extra parameter:

```
servicecatalog-puppet bootstrap-spoke <ACCOUNT_ID_OF_YOUR_PUPPET> --permission-
↪boundary arn:aws:iam::aws:policy/AdministratorAccess
```

The parameter is available on *bootstrap-spoke-as* and *bootstrap-spokes-in-ou* also.

Note: permission boundary was added in version 0.56.0

4.4 Customising the PuppetRole

When you bootstrap a spoke you can specify an optional *puppet-role-name* and *puppet-role-path*:

```
servicecatalog-puppet bootstrap-spoke <ACCOUNT_ID_OF_YOUR_PUPPET> --puppet-role-name␣
↪PuppetExecutionRole --puppet-role-path /automation/
```

Please note you must choose the same PuppetRoleName and PuppetRolePath for each spoke in your environment and you must choose that same PuppetRoleName and PuppetRolePath when bootstrapping the hub account:

You can have two independent hub accounts and bootstrap a spoke twice so that it can be managed by two independent hub accounts so long as the PuppetRoleName and PuppetRolePath are different.

Note: configurable puppet role name and paths were added in version 0.91.0

CHAPTER

FIVE

SHARING A PORTFOLIO

5.1 What is sharing and how does it work?

Note: This was added in version 0.1.14

This framework allows you to create portfolios in other accounts that mirror the portfolio in your hub account. The framework will create the portfolio for you and either copy or import the products (along with their versions) from your hub account into the newly created portfolio.

In addition to this, you can specify associations for the created portfolio and add launch constraints for the products.

Warning: Once a hub product version has been copied into a spoke portfolio it will not be updated.

5.2 How can I set it up?

The following is an example of how to add the portfolio `example-simple-central-it-team-portfolio` to all spokes tagged `scope:spoke`:

```
spoke-local-portfolios:
  account-vending-for-spokes:
    portfolio: example-simple-central-it-team-portfolio
    product_generation_method: copy
    depends_on:
      - account-iam-for-spokes
    associations:
      - arn:aws:iam::${AWS::AccountId}:role/MyServiceCatalogAdminRole
    constraints:
      launch:
        - product: account-vending-account-creation-shared
          roles:
            - arn:aws:iam::${AWS::AccountId}:role/MyServiceCatalogAdminRole
    deploy_to:
      tags:
        - tag: scope:spoke
          regions: default_region
```

The example above will create the portfolio once the `depends_on` launches have completed successfully.

The valid values for regions are: - enabled - this will deploy to each enabled region for the account - regions_enabled - this will deploy to each enabled region for the account - default_region - this will deploy to the default region specified for the account - all - this will deploy to all regions enabled in your config (whilst setting up Puppet) - list of AWS regions - you can type in a list of AWS regions (each region selected should be present in your config)

5.2.1 What are the settings for product_generation_method?

Note: Being able to configure this was added in version 0.72.0, previous versions always assume a copy

Using `spoke-local-portfolios`, a spoke portfolio is always created as a copy of the hub portfolio - that is, it has a different portfolio ID, but the same name and metadata as the hub portfolio. This ensures changes made to the portfolio in the spoke (such as associations and constraints) cannot affect other spokes or the hub portfolio.

There are 2 options for populating products inside the spoke portfolios, Copy or Import.

`product_generation_method: copy` (the default) means that products and provisioning artifacts are deep-copied into the spoke local portfolio using the Service Catalog `CopyProduct` API call.
They will get new product and provisioning artifact IDs, but have the same names and metadata as hub products.
This ensures isolation between hub and spoke - changes to products and versions in the hub will only be reflected in the spoke on the next puppet run.

`product_generation_method: import` setting will import (using the `AssociateProductWithPortfolio` Service Catalog API call) the product and its provisioning artifacts into the spoke from the hub portfolio with the same IDs as the hub portfolio.
This is useful for occasions where is it necessary to have the same product ID in the spoke as in the hub - for example if changes to provisioning artifacts in the hub need to be instantly reflected in the spoke without requiring a puppet run.

5.2.2 How can I add an association?

The example above will add an association for the IAM principal:

`arn:aws:iam::${AWS::AccountId}:role/MyServiceCatalogAdminRole`

so the portfolio will be accessible for anyone assuming that role. In addition to roles, you can also specify the ARN of users and groups.

Note: Using `${AWS::AccountId}` will evaluate in the spoke account.

5.2.3 How can I add a launch constraint?

The example above will add a launch constraint for the IAM role:

`arn:aws:iam::${AWS::AccountId}:role/MyServiceCatalogAdminRole`

so they can launch the product `account-vending-account-creation-shared` in the spoke account.

Warning: You can only specify an IAM role and the role must be assumable by the AWS service principal `servicecatalog.amazonaws.com`

Note: Using `${AWS::AccountId}` will evaluate in the spoke account.

Note: Support for using `products` was added in version 0.3.0.

You can use `products` instead of `product` to specify either a list of products or use a regular expression. The regular expression is matched using Python3 `re.match`.

Using a list:

```
spoke-local-portfolios:
  account-vending-for-spokes:
    portfolio: example-simple-central-it-team-portfolio
    depends_on:
      - account-iam-for-spokes
    associations:
      - arn:aws:iam::${AWS::AccountId}:role/MyServiceCatalogAdminRole
    constraints:
      launch:
        - products:
            - account-vending-account-bootstrap-shared
            - account-vending-account-creation-shared
          roles:
            - arn:aws:iam::${AWS::AccountId}:role/MyServiceCatalogAdminRole
    deploy_to:
      tags:
        - tag: scope:spoke
          regions: default_region
```

Using a regular expression:

```
spoke-local-portfolios:
  account-vending-for-spokes:
    portfolio: example-simple-central-it-team-portfolio
    depends_on:
      - account-iam-for-spokes
    associations:
      - arn:aws:iam::${AWS::AccountId}:role/MyServiceCatalogAdminRole
    constraints:
      launch:
        - products: "account-vending-account-*"
          roles:
            - arn:aws:iam::${AWS::AccountId}:role/MyServiceCatalogAdminRole
    deploy_to:
      tags:
        - tag: scope:spoke
          regions: default_region
```

5.2.4 How can I unshare a portfolio?

You can set the status on a spoke-local-portfolio. When you set it to terminated the spoke-local-portfolio is 'terminated'.

If the account hosting the spoke-local-portfolio is not the puppet account it will have the associations and constraints removed, the local portfolio will be deleted and the share with the puppet account will be deleted.

If the account hosting the spoke-local-portfolio is the puppet account it will delete the associations and the constraints but will leave the portfolio in place and will have no share to delete.

```
spoke-local-portfolios:
  account-vending-for-spokes:
    portfolio: example-simple-central-it-team-portfolio
    status: terminated
    depends_on:
      - account-iam-for-spokes
    associations:
      - arn:aws:iam::${AWS::AccountId}:role/MyServiceCatalogAdminRole
    constraints:
      launch:
        - products: "account-vending-account-*"
          roles:
            - arn:aws:iam::${AWS::AccountId}:role/MyServiceCatalogAdminRole
    deploy_to:
      tags:
        - tag: scope:spoke
          regions: default_region
```

Note: This was added in version 0.73.0

5.2.5 Sharing mode

Note: This was added in version 0.88.0

When you write a spoke-local-portfolio, the framework will share the portfolio used with each spoke account you are deploying into. The framework shares with each account and accepts the share within each account. You can tell the framework to share with an OU (using Organizational sharing) instead and then accept the share from within each account still. This reduces the time taken to share portfolios but means all accounts in the same OU will have the portfolio shared with them - those account will not have the portfolio share accepted. To enable this behaviour you need to set the sharing_mode:

```
spoke-local-portfolios:
  account-vending-for-spokes:
    portfolio: example-simple-central-it-team-portfolio
    sharing_mode: AWS_ORGANIZATIONS
    depends_on:
      - account-iam-for-spokes
    associations:
      - arn:aws:iam::${AWS::AccountId}:role/MyServiceCatalogAdminRole
    constraints:
      launch:
```

(continues on next page)

(continued from previous page)

```
        - products: "account-vending-account-*"
          roles:
            - arn:aws:iam::${AWS::AccountId}:role/MyServiceCatalogAdminRole
    deploy_to:
      tags:
        - tag: scope:spoke
          regions: default_region
```

To revert back you can set sharing_mode back to ACCOUNT:

```
spoke-local-portfolios:
  account-vending-for-spokes:
    portfolio: example-simple-central-it-team-portfolio
    sharing_mode: ACCOUNT
    depends_on:
      - account-iam-for-spokes
    associations:
      - arn:aws:iam::${AWS::AccountId}:role/MyServiceCatalogAdminRole
    constraints:
      launch:
        - products: "account-vending-account-*"
          roles:
            - arn:aws:iam::${AWS::AccountId}:role/MyServiceCatalogAdminRole
    deploy_to:
      tags:
        - tag: scope:spoke
          regions: default_region
```

If you are using this feature you must be able to share using Organizations in your puppet account. To do this you must have installed puppet into your AWS Organizations management account or you must have delegated your puppet account as an AWS Service Catalog organizations master account.

The default value for sharing_mode is ACCOUNT unless you change it using the following command:

```
servicecatalog-puppet set-named-config-value global_sharing_mode_default AWS_
↪ORGANIZATIONS
```

Alternatively, you can also add the following to your config:

```
global_sharing_mode_default: AWS_ORGANIZATIONS
```

When you change the global_sharing_mode_default it affects launches and spoke-local-portfolios.

5.3 What is the recommended implementation pattern?

1. Add an entry to launches that will provision a product into to your matching spokes. This product should provide the IAM roles your users will assume to interact with the portfolio you are going to add.
2. Add an entry to spoke-local-portfolios to add a portfolio to your matching spokes. This should depend on the product you launched that contains the IAM roles you added to the launches section of your manifest.

5.4 Is there anything else I should know?

1. It would be good to become familar with the AWS Service Catalog pricing before using this feature.

CHAPTER

SIX

EXECUTION MODES

6.1 What are execution modes

Execution modes alter the way the framework runs. When you change the value of the execution mode you change how or where the execution will occur.

Up until version 0.76.0 this framework only supported a single execution mode - hub. This meant there was a single pipeline provisioning products and sharing portfolios. This is still the default mode. When you configure a launch you can set the execution mode. If you do not set it, it will default back to hub. Read the following to understand more about the different execution modes.

6.2 How to set the execution mode

You can configure the execution mode for a launch by setting it:

```
IAM-1:
  portfolio: e-mandatory
  product: aws-iam-administrator-access-assumable-role-account
  version: v1
  execution: async
  parameters:
    AccountToTrust:
      default: '0123456789010'
    RoleName:
      default: 'SuperAdmin'
    Path:
      default: '/'
  deploy_to:
    tags:
      - tag: role:spokes
        regions: default_region
```

You can change the execution mode to any of the accepted values at any time.

6.3 Hub

Hub is the default. It means all provisioning occurs in the main AWS CodeBuild project in the puppet account. During a hub execution the CodeBuild project will wait for each execution to complete before processing its dependents.

6.4 Async

Note: This was added in version 0.76.0

With async the provisioning still occurs in the CodeBuild project of the puppet account but when provisioning of launches is running the CodeBuild project does not wait for the completion of the product provisioning. This means you cannot depend on a launch that is async and you cannot have outputs for an async launch.

6.5 Spoke

Note: This was added in version 0.78.0

Note: This will require you to bootstrap your spokes again if you bootstrapped your spokes with a version prior to 0.78.0 - which was July 8, 2020. You can check the version of the spoke in AWS SSM Parameter store (a parameter named service-catalog-puppet-spoke-version in your home region should be present). If you need to bootstrap your spokes again you can bootstrap each in the same way you initially did it otherwise you can use the AWS Codebuild project `servicecatalog-puppet-bootstrap-spokes-in-ou`

With spoke the provisioning starts in the puppet account as normal. Once a launch with spoke execution mode is found an AWS Codebuild project is triggered in the spoke account to perform the provisioning. The framework will run the project only once per account per run of the main puppet run to avoid wasteful execution times. The depends_on statements are adhered to within the same account but depending on launches across accounts are not adhered to. When using AWS SSM parameters in spoke execution mode you will need to declare which account the SSM parameter is available in. You can use ${AWS::PuppetAccountId} or ${AWS::AccountId} to specific the hub or the spoke. If you specify the hub account the SSM parameter will be retrieved and its value will be shared with the spoke. When sharing from the hub to the spoke there is no cross account permission added, the values of the parameters are shared in text files with the spoke account AWS CodeBuild project.

Here is an example using a parameter in the hub account to share values with the spoke.

```
IAM-1:
  portfolio: e-mandatory
  product: aws-iam-administrator-access-assumable-role-account
  version: v1
  execution: spoke
  parameters:
    AccountToTrust:
      ssm:
        name: '/accounts/tooling/accountId'
        account_id: '${AWS::PuppetAccountId}'
    RoleName:
      default: 'SuperAdmin'
    Path:
      default: '/'
  deploy_to:
    tags:
      - tag: role:spokes
        regions: default_region
```

CHAPTER

SEVEN

NOTIFICATIONS

You can listen to the AWS CloudFormation stack events from your product provisioning.

This is the recommended way of discovering provisioning errors.

When you bootstraped your account you will have created an AWS SQS Queue: `servicecatalog-puppet-cloudformation-events` in your default region.

You will also have SNS Topics in each region configured to push events to this queue: `servicecatalog-puppet-cloudformation-regional-events`

Please note this will only receive notifications for products provisioned using ServiceCatalog-Puppet - any self service vending from AWS Service Catalog will not publish to this queue.

You should handle consuming the queue and have your own error handling code.

CHAPTER

EIGHT

CACHING

8.1 What is caching and how does it work?

Note: This was added in version 0.84.0

This framework uses the python library Luigi to generate and execute its workflow and to handle dependencies. This framework generates a series of tasks that will provision products, share portfolios and execute lambda functions. Luigi then starts workers that will consume the tasks and report back on their status.

The tasks we generate are fairly fine grained to enabled better concurrency and to make use of Luigi's ability to cache task instances so that the exact same task will not be executed more than once.

Some of these tasks we generate in this framework should never need to be executed more than once - for example getting the provisioning parameters for a specific AWS Service Catalog provisioning artifact.

When Luigi runs a task it generates a target which is a file containing the result of the task. Each time Luigi runs it checks to see if the target exists already - if it does then it will not rerun the task.

You can enable the caching of the following tasks:

- DeleteCloudFormationStackTask
- EnsureEventBridgeEventBusTask
- BootstrapSpokeAsTask
- ShareAndAcceptPortfolioTask
- CreateAssociationsInPythonForPortfolioTask
- CreateShareForAccountLaunchRegion
- DoDescribeProvisioningParameters

Warning: Since 0.84.0 this feature has only been tested when execution mode is hub. If you encounter any issues in any other mode please raise an issue. Support for other execution modes will be added shortly.

8.2 What should I do if I think I see an issue?

Please check through the github issues to check if your issue has been raised already. If not, please raise a new github issue so progress can be tracked.

8.3 How can I empty the cache?

The cache is copied over to AWS S3 after every run and copied back before each future run. You can delete the cache should you want to empty it:

```
aws s3 rm --recursive s3://sc-puppet-caching-bucket-${PUPPET_ACCOUNT_ID}-${AWS_REGION}
↪/output
```

It is not recommended to delete specific files from the cache as there is a complex relationship between the files due to the task dependencies.

8.4 How do I enable this?

In order to use this feature you will need to enable caching using the CLI:

```
servicecatalog-puppet set-config-value is_caching_enabled True
```

8.5 How do I disable this?

In order to use disable feature you will need to disable caching using the CLI:

```
servicecatalog-puppet set-config-value is_caching_enabled False
```

CHAPTER

NINE

INTRINSIC FUNCTIONS

9.1 What are intrinsic functions

Note: The first intrinsic function was added in version 0.115.0

Intrinsic functions allow you to specify a token in the manifest.yaml file and have it replaced with a value generated by the tools at runtime. The aim of them is to make your manifest.yaml file more portable across environments and to make your life managing the manifest.yaml file easier.

9.1.1 PuppetAccountId

Note: This was added in version 0.115.0

You can use the token ${AWS::PuppetAccountId} anywhere in your manifest files. When you do so it will be replaced with the puppet account id you are using. This replacement occurs in the expand phase and the deploy phase is unaware of the notation. When using the PuppetAccountId intrinsic function we recommend ensuring it is string by surrounding it in double quotes - this avoids issues where the account id may be passed as a number later on.

Here is an example usage of it in the parameters section:

```
schema: puppet-2019-04-01
parameters:
  PuppetAccountId1:
    default: "${AWS::PuppetAccountId}"
  OrganizationAccountAccessRole:
    default: OrganizationAccountAccessRole
```

And here is an example in a stack:

```
stacks:
  networking-stack:
    name: networking-stack
    version: v1
    parameters:
      PuppetAccountId:
        default: "${AWS::PuppetAccountId}"
    deploy_to:
      tags:
```

(continues on next page)

(continued from previous page)

```
      - regions: default_region
        tag: role:puppethub
```

CHAPTER

TEN

SPLITTING UP THE WORKFLOW

10.1 How does ServiceCatalogPuppetWork

Service Catalog Puppet converts your manifest file into a set of tasks that are subsequently run in a workflow that is managed by Luigi.

When you use depends_on within your manifest file you are declaring that a set of tasks now depend on another set. This is creating sub workflows.

By default, every time Service Catalog Puppet runs it will try to execute each of the tasks in each of the sub workflows, it does not even see any of the sub workflows, it just schedules the tasks as if there is one workflow.

10.2 What problems can this cause?

By design, the Service Catalog Tools are developed to start simple and only increase in complexity when needed.

When the number of launches, spoke-local-portfolios, lambda executions, actions, regions or accounts grow to certain size you may find that you want to optimise the workflow to speed it up. You can do so by using AWS Orgs sharing, task level caching or by optimising the way you build your AWS Service Catalog products or populate their parameters.

Eventually, you will most likely want to split your single pipeline run into smaller runs and only run pipelines handling changes that needed instead of verifying all changes are still in place.

10.3 How can I fix this?

Since version 0.99.0 there has been a second source for the servicecatalog-puppet-pipeline named Parameterised-Source. This is an AWS S3 bucket which you can place a parameter file in. When you place the parameter file in the bucket it will trigger the pipeline. The pipeline will use the latest version of your ServiceCatalogPuppet git repo for your manifest file and will use the parameters you placed in the file to trigger a special run of the pipeline.

The options you specify must be placed in a parameters.yaml file, in a parameters.zip file in the root of your parameters bucket.

Using the parameters file you can fine tune the pipeline to run a subset of the workflow by specifying one or more of the following options:

10.3.1 Running the pipeline for a single account

Note: This was added in version 0.99.0

You can use the following attribute

```
single_account: ${SINGLE_ACCOUNT_ID}
```

10.3.2 Running the pipeline for a subset of operations

Note: This was added in version 0.101.0 and the syntax changed in 0.165.0 but backwards compatability is maintained

You can use the following attribute:

```
item: "remove-default-vpc-function"
section: "launches"
include_dependencies: true
include_reverse_dependencies: true
```

item and section correspond to which operation you want to run in your pipeline. Here we are saying run the launch named remove-default-vpc-function.

include_dependencies is optional and defaults to false. When set to false it does nothing. When set to true if will ensure that anything remove-default-vpc-function depends_on is included also.

include_reverse_dependencies is optional and defaults to false. When set to false it does nothing. When set to true if will ensure that anything depending on remove-default-vpc-function is included also.

10.3.3 Recommended use

It is recommended you use this only if you are unable to operate with the full pipeline run. When using this you are managing the state yourself and it is easy to miss operations. If you are using this we recommend running a regular nightly or suitable other time based run to ensure you do not miss operations, regions or accounts.

When using this we recommend you turn off polling for source changes on your ServiceCatalogPuppet repo - this can be done via the bootstrap command. If you decide to stop using this you can turn on polling again and you back to how you started.

CHAPTER

ELEVEN

FREQUENTLY ASKED QUESTIONS (FAQ)

11.1 PuppetRole has been recreated

Q. My PuppetRole has been recreated and now I cannot perform updates to provisioned products. What should I do?

A. You will need to follow these steps:

- Delete the AWS Cloudformation Stacks named `servicecatalog-puppet-shares`. There will be one in each region you operate in. you can use the utility delete-stack-from-all-regions to help
- Run the puppet pipeline again
- Run the cli command reset-provisioned-product-owner on your expanded manifest file.

11.2 How do I enable OpsCenter support

Q. How do I enable OpsCenter support?

A. You will need to be running at least version 0.35.0. You can check your version by running the version cli command. If it is below 0.35.0 you will need to upgrade. Once you are running the correct version you will been to update your config file to include:

```
should_forward_failures_to_opscenter: true
```

Your file should look like the following:

```
regions: [
  'eu-west-1',
  'eu-west-2',
  'eu-west-3'
]
should_forward_failures_to_opscenter: true
```

Once you have made the change you will need to upload your config again:

```
servicecatalog-puppet upload-config config.yaml
```

11.3 How can I exclude an account or a sub Organizational unit from an expand

Q. How can I exclude an account or a sub Organizational unit from an expand?

A. You can use an exclude attribute within your ou block:

```
- ou: /
  default_region: eu-west-1
  name: '665532578041'
  regions_enabled:
    - eu-west-1
    - eu-west-2
  tags:
    - type:spoke
    - partition:eu
    - scope:pci
  exclude:
    accounts:
      - 0123456789010
    ous:
      - '/hub'
```

Excludes can include accounts or ous. Please note the ous work recursively so the example above to create a list of accounts that is all accounts in the organization excluding account `0123456789010` and excluding all accounts from `/hub` and all of `/hub` sub organizations

Note: exclude was added in version 0.53.0

11.4 How can I add tags or regions or change the default region of a single account within an OU

Q. How can I add tags or regions or change the default region of a single account within an OU?

A. You can continue using the OU notation to describe the group of accounts the account you want to customise sits in:

```
- ou: /eu-dev
  name: 'eu-dev'
  default_region: eu-west-1
  regions_enabled:
    - eu-west-1
    - eu-west-2
  tags:
    - type:spoke
    - partition:eu
```

You can then add a tag to the account by adding the following to your manifest:

```
- account_id: '665532578041'
  name: '665532578041'
  append:
    tags:
      - scope:pci
```

This means all accounts in the eu-dev OU will have the following tags:

```
- type:spoke
- partition:eu
```

and account '665532578041' will have the following tags:

```
- type:spoke
- partition:eu
- scope:pci
```

Instead of using append you can also use overwrite. Using overwrite would replace the tags in the example above.

When using append you can specify tags and regions_enabled to be appended.

When using overwrite you can specify default_region, tags and regions_enabled to be overwritten.

Note: exclude was added in version 0.63.0

CHAPTER

TWELVE

KITCHEN SINK EXAMPLE

Here is an example manifest showing how to use the capabilities of the tools:

```
accounts:
  # defining an account
  - account_id: &service_catalog_tools_account '234982635846243'
    name: 'service_catalog_tools_account'
    default_region: &default_region eu-west-1
    regions_enabled: &regions_enabled
      - eu-west-1
      - eu-west-2
      - eu-west-3
      - us-east-1
      - us-east-2
      - us-west-1
      - us-west-2
    tags:
      - &outype_foundational outype:foundational
      - &ou_sharedservices ou:sharedservices
      - &partition_eu partition:eu
      - &partition_us partition:us
      - &role_hub role:hub
      - &role_service_catalog_tools role:service_catalog_tools
      - &team_ccoe team:ccoe

  - account_id: '9832654846594385'
    name: 'org-manager'
    default_region: us-east-1
    regions_enabled:
      - us-east-1
    tags:
      # using yaml anchors and aliases from above
      - *outype_foundational
      - *ou_sharedservices
      - *partition_us
      - *role_hub
      - &role_org_manager role:org_manager
      - *team_ccoe

  # defining each account in the OU
  - ou: '/workloads/test'
    name: 'workloads-test'
    default_region: *default_region
    regions_enabled: *regions_enabled
    tags:
```

(continues on next page)

(continued from previous page)

```
      - &outype_additional outype:additional
      - &outype_workloads outype:workloads
      - &outype_test outype:test
      - *partition_us
      - *partition_eu
      - &role_spoke role:spoke
    # excluding an account managed by another team
    exclude:
      accounts:
        - "07632469093733"

  # this is a test account but contains PCI data as it is used for load testing and␣
↪needs real data
  - account_id: '665532578041'
    name: '665532578041'
    # add the tag
    append:
      tags:
        - &score_pci scope:pci

  # this was a test account but is now the active directory account and we cannot␣
↪move it to the correct ou
  - account_id: '30972093754'
    name: 'active-directory'
    # overwrite the tags
    overwrite:
      tags:
        - *outype_foundational
        - *ou_sharedservices
        - *partition_us
        - *partition_eu
        - *role_hub

# define some global parameters for provisioning products in launches below
parameters:
  # Whenever a product has a parameter the framework will use the parameters␣
↪specified in the accounts section first,
  # then the launch itself and then finally the global
  ServiceCatalogToolsAccountId:
    default: *service_catalog_tools_account

  # the framework will execute the function get_accounts_for_path with the args / and␣
↪then use the result for the value
  # of this parameter
  AllAccountIds:
    macro:
      method: get_accounts_for_path
      args: /


# define some mappings that can be used as parameters for launches.  mappings allow␣
↪us to define groups of parameters
# just like cloudformation does
mappings:
  InternetGatewayDeviceAMI:
    us-east-1:
      "ami": "ami-15f77f867"
```

(continues on next page)

(continued from previous page)

```
    us-west-1:
      "ami": "ami-0bdb82235"
    eu-west-1:
      "ami": "ami-16506cd98"

# actions are wrappers around codebuild projects.  they allow you to run the project␣
↪and only continue exection
# should the project be successful
actions:
  ping-on-prem-host:
    type: codebuild
    project_name: &ping_on_prem_host ping-on-prem-host
    account_id: *service_catalog_tools_account
    region: 'eu-west-1'
    parameters:
      HOST_TO_PING:
        default: 192.168.1.2


launches:
  # provision v1 of account-bootstrap-shared-org-bootstrap from demo-central-it-team-
↪portfolio into the role_org_manager
  # account
  account-bootstrap-shared-org-bootstrap:
    portfolio: demo-central-it-team-portfolio
    product: account-bootstrap-shared-org-bootstrap
    version: v1
    parameters:
      # Use some parameters for the provisioning
      GovernanceAtScaleAccountFactoryAccountBootstrapSharedBootstrapperOrgIAMRoleName:
        default: AccountBootstrapSharedBootstrapperOrgIAMRoleName
      GovernanceAtScaleAccountFactoryIAMRolePath:
        default: /AccountFactoryIAMRolePath/
      OrganizationAccountAccessRole:
        default: OrganizationAccountAccessRole
    deploy_to:
      tags:
        # deploy only to the default region - which can be a different region per␣
↪account
        - regions: default_region
          tag: *role_org_manager
    # Store the output from the provisioned product / cloudformation stack in SSM (in␣
↪the service catalog tools account)
    outputs:
      ssm:
        - param_name: &AssumableRoleArnInRootAccountForBootstrapping /governance-at-
↪scale-account-factory/account-bootstrap-shared-org-bootstrap/
↪AssumableRoleArnInRootAccountForBootstrapping
          stack_output: AssumableRoleArnInRootAccountForBootstrapping

  account-bootstrap-shared:
    portfolio: demo-central-it-team-portfolio
    product: account-bootstrap-shared
    version: v2
    parameters:
      AssumableRoleArnInRootAccountForBootstrapping:
        # use a parameter from SSM (in the service catalog tools account)
```

(continues on next page)

(continued from previous page)

```
        ssm:
          name: *AssumableRoleArnInRootAccountForBootstrapping
      GovernanceAtScaleAccountFactoryAccountBootstrapSharedBootstrapperIAMRoleName:
        default: AccountBootstrapSharedBootstrapperIAMRoleName
      GovernanceAtScaleAccountFactoryAccountBootstrapSharedCustomResourceIAMRoleName:
        default: AccountBootstrapSharedCustomResourceIAMRoleName
      GovernanceAtScaleAccountFactoryIAMRolePath:
        default: /AccountFactoryIAMRolePath/
    # only provision this if account-bootstrap-shared-org-bootstrap provisions␣
↪correctly
    depends_on:
      - account-bootstrap-shared-org-bootstrap
    outputs:
      ssm:
        - param_name: &
↪GovernanceAtScaleAccountFactoryBootstrapperProjectCustomResourceArn /governance-at-
↪scale-account-factory/account-bootstrap-shared/
↪GovernanceAtScaleAccountFactoryBootstrapperProjectCustomResourceArn
          stack_output:␣
↪GovernanceAtScaleAccountFactoryBootstrapperProjectCustomResourceArn
    deploy_to:
      tags:
        - regions: default_region
          tag: *role_service_catalog_tools

  internet-gateway:
    portfolio: networking
    product: internet-gateway
    version: v3
    deploy_to:
      tags:
        # regions can also be a list
        - regions:
            - us-east-1
            - us-west-1
            - eu-west-1
          tag: *role_spoke
    parameters:
      AMI:
        # use a mapping as a parameter. when provisioning occurs AWS::Region is␣
↪replaced with the region being
        # provisioned so you can create region specified parameters in the manifest␣
↪file, you can also use
        # AWS::AccountId to create account specific parameters in the manifest file
        mapping: [InternetGatewayDeviceAMI, AWS::Region, ami]

  vpc:
    portfolio: networking
    product: vpc
    version: v8
    # before provisioning this product into the specified accounts run the pre_action.
↪  If that project fails this
    # launch will not be provisioned
    pre_actions:
      - name: *ping_on_prem_host
    deploy_to:
      tags:
```

(continues on next page)

(continued from previous page)

```
        - regions:
            - us-east-1
            - us-west-1
            - eu-west-1
          tag: *role_spoke
    parameters:
      NetworkType:
        ssm:
          # when the framework is getting the ssm parameter (in the service catalog tools account) you can use
          # ${AWS::AccountId} and ${AWS::Region} in the name to build out a name dynamically allowing you to use
          # SSM parameter store as a data store for the configuration of each account
          name: /networking/vpc/account-parameters/${AWS::AccountId}/${AWS::Region}/NetworkType
      CIDR:
        ssm:
          name: /networking/vpc/account-parameters/${AWS::AccountId}/${AWS::Region}/CIDR
    outputs:
      ssm:
        # You can also use ${AWS::AccountId} and ${AWS::Region} in the output parameter name
        - param_name: /networking/vpc/account-parameters/${AWS::AccountId}/${AWS::Region}/VPCId
          stack_output: VPCId

  remove-default-vpc-lambda:
    portfolio: networking
    product: remove-default-vpc-lambda
    version: v3
    parameters:
      RemoveDefaultVPCFunctionName:
        default: &RemoveDefaultVPCFunctionName RemoveDefaultVPC
    deploy_to:
      tags:
        - regions: *default_region
          tag: *service_catalog_tools_account


lambda-invocations:
  # this lambda is executed in the service catalog tools account for each region of each account defined in the
  # invoke_for.  The values of account_id and region are available as parameters to the lambda.
  remove-default-vpc:
    function_name: *RemoveDefaultVPCFunctionName
    qualifier: $LATEST
    invocation_type: Event
    # wait until the lambda is provisioned as part of the launch
    depends_on:
      - name: remove-default-vpc-lambda
        type: launch
    invoke_for:
      tags:
        - regions:
            - us-east-1
```

(continues on next page)

(continued from previous page)

```
            - us-west-1
            - eu-west-1
          tag: *role_spoke


spoke-local-portfolios:
  networking-self-service:
    portfolio: networking-self-service
    # import the product and not copy it
    product_generation_method: import
    associations:
      - arn:aws:iam::${AWS::AccountId}:role/ServiceCatalogConsumer
    constraints:
      launch:
        - product: account-vending-account-creation-shared
          roles:
            - arn:aws:iam::${AWS::AccountId}:role/ServiceCatalogProvisioner
    deploy_to:
      tags:
        - tag: *role_spoke
          regions: default_region
```

CHAPTER

THIRTEEN

USING THE CLI

The following utils will help you manage your AWS Accounts when using ServiceCatalog-Puppet:

13.1 reset-provisioned-product-owner

Note: This was added in version 0.19.0

You can use the `servicecatalog-puppet` cli to update the Service Catalog Puppet managed provisioned product owner for each provisioned product across all of your accounts:

```
servicecatalog-puppet reset-provisioned-product-owner <path_to_expanded_manifest>
```

Will call the following function for each provisioned product you have:

```
service_catalog.update_provisioned_product_properties(
    ProvisionedProductId=provisioned_product_id,
    ProvisionedProductProperties={
        'OWNER': f"arn:aws:iam::{self.account_id}:role/servicecatalog-puppet/
↪PuppetRole"
    }
)
```

13.2 add-to-accounts

Note: This was added in version 0.18.0

You can use the `servicecatalog-puppet` cli to see add an account or ou to your accounts list:

```
servicecatalog-puppet add-to-accounts <path_to_file_containing_account_or_ou>
```

The file containing the account or ou should be structured like this:

```
account_id: '<AccountID>'
default_region: eu-west-1
name: '<AccountID>'
regions_enabled:
```

(continues on next page)

(continued from previous page)

```
  - eu-west-1
  - eu-west-2
tags:
  - type:prod
  - partition:eu
  - scope:pci
```

13.3 remove-from-accounts

Note: This was added in version 0.18.0

You can use the `servicecatalog-puppet` cli to remove an account or ou to your accounts list:

```
servicecatalog-puppet remove-from-accounts <account_id_or_ou_id_or_ou_path>
```

The library will look for the given account id, ou id or ou path and remove it, if found. If it is missing an exception will be raised.

13.4 add-to-launches

Note: This was added in version 0.18.0

You can use the `servicecatalog-puppet` cli to see add a launch to your launches list:

```
servicecatalog-puppet add-to-launches <launch-name-to-add> <path_to_file_containing_
↪launch>
```

The file containing the launch should be structured like this:

```
portfolio: example-simple-central-it-team-portfolio
product: aws-iam-assume-roles-spoke
version: v1
parameters:
  SecurityAccountId:
    default: '<AccountID>'
deploy_to:
  tags:
    - regions: default_region
      tag: type:prod
```

13.5 remove-from-launches

Note: This was added in version 0.18.0

You can use the `servicecatalog-puppet` cli to see remove a launch from your launches list:

```
servicecatalog-puppet remove-from-launches <launch-name-to-remove>
```

13.6 dry-run

Note: This was added in version 0.8.0

You can use the `servicecatalog-puppet` cli to see the effect of your next pipeline run before it happens

```
servicecatalog-puppet dry-run ServiceCatalogPuppet/manifest.yaml
```

You must specify the path to the manifest file you want to add execute a dry run on.

13.7 import-product-set

Note: This was added in version 0.8.0

You can use the `servicecatalog-puppet` cli to import products from the aws-service-catalog-products shared repo.

This will update your manifest file.

```
servicecatalog-puppet import-product-set ServiceCatalogPuppet/manifest.yaml aws-iam␣
↪central-it-team-portfolio
```

You must specify the path to the manifest file you want to add the product set to, the name of the product set and the name of the portfolio where was added.

13.8 list-resources

Note: This was added in version 0.7.0

You can use the `servicecatalog-puppet` cli to list all the resources that will be created to bootstrap the framework

```
servicecatalog-puppet list-resources
```

Will return the following markdown:

```
# Framework resources
## SSM Parameters used
- /servicecatalog-puppet/config
## Resources for stack: servicecatalog-puppet-org-master

| Logical Name            | Resource Type        | Name                                ␣
↪       |
```

(continues on next page)

(continued from previous page)

```
| Param                    | AWS::SSM::Parameter | service-catalog-puppet-org-master-version |
| PuppetOrgRoleForExpands  | AWS::IAM::Role      | PuppetOrgRoleForExpands                   |

## Resources for stack: servicecatalog-puppet-regional

| Logical Name             | Resource Type       | Name                                                                                  |
| DefaultRegionParam       | AWS::SSM::Parameter | /servicecatalog-puppet/home-region                                                    |
| Param                    | AWS::SSM::Parameter | service-catalog-puppet-regional-version                                               |
| PipelineArtifactBucket   | AWS::S3::Bucket     | Fn::Sub: sc-puppet-pipeline-artifacts-${AWS::AccountId}-${AWS::Region}                |
| RegionalProductTopic     | AWS::SNS::Topic     | servicecatalog-puppet-cloudformation-regional-events                                  |

## Resources for stack: servicecatalog-puppet-spoke

| Logical Name | Resource Type       | Name                                |
| Param        | AWS::SSM::Parameter | service-catalog-puppet-spoke-version |
| PuppetRole   | AWS::IAM::Role      | PuppetRole                          |

## Resources for stack: servicecatalog-puppet

| Logical Name                  | Resource Type                | Name                                                                                  |
| Param                         | AWS::SSM::Parameter          | service-catalog-puppet-version                                                        |
| ShareAcceptFunctionRole       | AWS::IAM::Role               | ShareAcceptFunctionRole                                                               |
| ProvisioningRole              | AWS::IAM::Role               | PuppetProvisioningRole                                                                |
| CloudFormationDeployRole      | AWS::IAM::Role               | CloudFormationDeployRole                                                              |
| PipelineRole                  | AWS::IAM::Role               | PuppetCodePipelineRole                                                                |
| SourceRole                    | AWS::IAM::Role               | PuppetSourceRole                                                                      |
| CodeRepo                      | AWS::CodeCommit::Repository  | ServiceCatalogPuppet                                                                  |
| Pipeline                      | AWS::CodePipeline::Pipeline  | Fn::Sub: ${AWS::StackName}-pipeline                                                   |
| GenerateRole                  | AWS::IAM::Role               | PuppetGenerateRole                                                                    |
| DeployRole                    | AWS::IAM::Role               | PuppetDeployRole                                                                      |
| GenerateSharesProject         | AWS::CodeBuild::Project      | servicecatalog-puppet-generate                                                        |
```

(continues on next page)

(continued from previous page)

```
| DeployProject                    | AWS::CodeBuild::Project      | servicecatalog-
↪puppet-deploy                   |
| SingleAccountRunProject          | AWS::CodeBuild::Project      | servicecatalog-
↪puppet-single-account-run       |
| CloudFormationEventsQueue        | AWS::SQS::Queue              | servicecatalog-
↪puppet-cloudformation-events    |
| CloudFormationEventsQueuePolicy | AWS::SQS::QueuePolicy        | -                ␣
↪                                |
└──────────────────────────────────┴──────────────────────────────┴──────────────────

n.b. AWS::StackName evaluates to servicecatalog-puppet
```

13.9 run

Note: This was added in version 0.3.0

The run command will run the main AWS CodePipeline `servicecatalog-puppet-pipeline`

```
servicecatalog-puppet run
```

You can also tail the command to watch the progress of the pipeline. It is a little underwhelming at the moment.

```
servicecatalog-puppet run --tail
```

13.10 list-launches

The list-launches command can currently only be invoked on an expanded manifest.yaml file. To expand your manifest you must run the following:

```
servicecatalog-puppet expand manifest.yaml
```

This will create a file named `manifest-expanded.yaml in the same directory`.

You can then run `list-launches`:

```
servicecatalog-puppet list-launches manifest-expanded.yaml
```

Here is an example table produced by running the command:

```
+--------------+-----------+------------------------------+--------------------------
↪---------------+----------------------------------+------------------+---------------
↪-+---------+-----------+
| account_id   | region    | launch                       | portfolio                ␣
↪               | product                          | expected_version | actual_
↪version | active | status    |
+--------------+-----------+------------------------------+--------------------------
↪---------------+----------------------------------+------------------+---------------
↪-+---------+-----------+
| 012345678901 | eu-west-1 | iam-assume-roles-spoke       | example-simple-central-it-
↪team-portfolio | aws-iam-assume-roles-spoke       | v1               | v1           ␣
↪ | True    | AVAILABLE |
```

(continues on next page)

(continued from previous page)

```
| 012345678901 | eu-west-1 | iam-groups-security-account  | example-simple-central-it-
↪team-portfolio | aws-iam-groups-security-account | v1               | v1
↪ | True   | AVAILABLE |
+--------------+-----------+------------------------------+--------------------------
↪---------------+----------------------------------+-----------------+---------------
↪-+--------+-----------+
```

Note: This was added in version 0.15.0

You can specify the format of the output. Currently you can choose between `json` and `table`. The default is table.

```
servicecatalog-puppet list-launches manifest-expanded.yaml --format json
```

13.11 export-puppet-pipeline-logs

The export-puppet-pipeline-logs takes a pipeline execution id and outputs a log file containing the AWS CloudWatch logs for each AWS CodeBuild step within the pipeline. This is useful for sharing these outputs when debugging:

```
servicecatalog-puppet export-puppet-pipeline-logs qwertyui-qwer-qwer-qwer-qwertyuiopqw
```

Note: This was added in version 0.47.0

13.12 graph

The graph command takes a task reference file as a parameter and generates a graphviz formated graph representing the actions the framework will perform

```
servicecatalog-puppet graph <path_to_expanded_manifest>
```

Note: This was added in version 0.49.0

13.13 Validate

The validate command will check your manifest file is of the correct structure and it will also ensure you are not using depends_on or deploy_to.tags that do not exist.

```
servicecatalog-puppet validate <path_to_manifest>
```

Note: This only works with the non expanded manifest file

13.14 show-codebuilds

Note: This was added in version 0.133.0

The show-codebuilds command will query the AWS CodeBuild APIs to gather the statistics for every execution of the AWS CodeBuild projects used by the solution.

```
servicecatalog-puppet show-codebuilds
```

The following options are available for the command:

–filter - you can filter for single-runs, full-runs or none –format - you can output as csv or json format –limit - you can limit the number of results you want to be returned

Note: If using Excel to analyze the data stored in CSV format, the "X Y" chart type provides a suitable visualization of build duration over time.

CHAPTER

FOURTEEN

USING THE SDK

Note: This was added in 0.18.0

Service Catalog Puppet includes a published SDK. You can make use of the python functions available:

```
from servicecatalog_puppet import sdk
```

The functions available are:

14.1 Functions

servicecatalog_puppet.sdk.**add_to_accounts**(*account_or_ou*)
Add the parameter to the account list of the manifest file

Parameters **account_or_ou** – A dict describing the the account or the ou to be added

servicecatalog_puppet.sdk.**add_to_launches**(*launch_name*, *launch*)
Add the given `launch` to the launches section using the given `launch_name`

Parameters

- **launch_name** – The launch name to use when adding the launch to the manifest launches
- **launch** – The dict to add to the launches

servicecatalog_puppet.sdk.**bootstrap**(*with_manual_approvals*, *puppet_account_id*, *puppet_code_pipeline_role_permission_boundary='arn:aws:iam::aws:policy/Admin*
source_role_permissions_boundary='arn:aws:iam::aws:policy/AdministratorAcc
puppet_generate_role_permission_boundary='arn:aws:iam::aws:policy/Adminis
puppet_deploy_role_permission_boundary='arn:aws:iam::aws:policy/Administr
puppet_provisioning_role_permissions_boundary='arn:aws:iam::aws:policy/Ad
cloud_formation_deploy_role_permissions_boundary='arn:aws:iam::aws:policy
deploy_environment_compute_type='BUILD_GENERAL1_SMALL', *deploy_num_workers=10*)
Bootstrap the puppet account. This will create the AWS CodeCommit repo containing the config and it will also create the AWS CodePipeline that will run the solution.

Parameters

- **with_manual_approvals** – Boolean to specify whether there should be manual approvals before provisioning occurs
- **puppet_account_id** – AWS Account Id for your puppet account

- **puppet_code_pipeline_role_permission_boundary** – IAM Boundary to apply to the role: PuppetCodePipelineRole
- **source_role_permissions_boundary** – IAM Boundary to apply to the role: SourceRole
- **puppet_generate_role_permission_boundary** – IAM Boundary to apply to the role: PuppetGenerateRole
- **puppet_deploy_role_permission_boundary** – IAM Boundary to apply to the role: PuppetDeployRole
- **puppet_provisioning_role_permissions_boundary** – IAM Boundary to apply to the role: PuppetProvisioningRole
- **cloud_formation_deploy_role_permissions_boundary** – IAM Boundary to apply to the role: CloudFormationDeployRole
- **deploy_environment_compute_type** – The AWS CodeBuild Environment Compute Type
- **deploy_num_workers** – Number of workers that should be used when running a deploy

servicecatalog_puppet.sdk.**bootstrap_spoke**(*puppet_account_id*, *permission_boundary*)

Bootstrap a spoke so that is can be used by the puppet account to share portfolios and provision products. This must be run in the spoke account.

Parameters

- **puppet_account_id** – this is the account id where you have installed aws-service-catalog-puppet
- **permission_boundary** – the iam boundary to apply to the puppetrole in the spoke account

servicecatalog_puppet.sdk.**bootstrap_spoke_as**(*puppet_account_id*, *iam_role_arns*, *permission_boundary*, *puppet_role_name='PuppetRole'*, *puppet_role_path='/servicecatalog-puppet/'*, *tag=[]*)

Bootstrap a spoke so that it can be used by the puppet account to share portfolios and provision products. This must be run in an account where you can assume the first ARN in the iam_role_arns list.

Parameters

- **puppet_account_id** – this is the account id where you have installed aws-service-catalog-puppet
- **iam_role_arns** – this is a list of ARNs the function will assume (in order) before bootstrapping. The final ARN in the list should be the ARN of the spoke you want to bootstrap.
- **permission_boundary** – the iam boundary to apply to the puppetrole in the spoke account

servicecatalog_puppet.sdk.**bootstrap_spokes_in_ou**(*ou_path_or_id*, *role_name*, *iam_role_arns*, *permission_boundary*, *num_workers=10*, *puppet_role_name='PuppetRole'*, *puppet_role_path='/servicecatalog-puppet/'*, *tag=[]*)

Bootstrap each spoke in the given path or id

Parameters

- **ou_path_or_id** – This is the ou path /example or the ou id for which you want each account bootstrapped
- **role_name** – This is the name (not ARN) of the IAM role to assume in each account when bootstrapping
- **iam_role_arns** – this is a list of ARNs the function will assume (in order) before bootstrapping. The final ARN in the list should be the ARN of account that can assume the role_name in the accounts to bootstrap.
- **permission_boundary** – the iam boundary to apply to the puppetrole in the spoke account

servicecatalog_puppet.sdk.**release_spoke**(*puppet_account_id*)
Delete the resources created during the bootstrap spoke process

Parameters **puppet_account_id** – AWS Account Id for your puppet account

servicecatalog_puppet.sdk.**remove_from_accounts**(*account_id_or_ou_id_or_ou_path*)
remove the given account_id_or_ou_id_or_ou_path from the account list

Parameters **account_id_or_ou_id_or_ou_path** – the value can be an account_id, ou_id or an ou_path. It should be present in the accounts list within the manifest file or an error is generated

servicecatalog_puppet.sdk.**remove_from_launches**(*launch_name*)
remove the given launch_name from the launches list

Parameters **launch_name** – The name of the launch to be removed from the launches section of the manifest file

servicecatalog_puppet.sdk.**run**(*what='puppet'*, *wait_for_completion=False*)
Run something

Parameters

- **what** – what should be run. The only parameter that will work is puppet
- **wait_for_completion** – Whether the command should wait for the completion of the pipeline before it returns

servicecatalog_puppet.sdk.**uninstall**(*puppet_account_id*)
Delete the resources created during the bootstrap process. AWS Service Catalog portfolios and their configurations are not modified during this call

Parameters **puppet_account_id** – AWS Account Id for your puppet account

servicecatalog_puppet.sdk.**upload_config**(*config*)
This function allows you to upload your configuration for puppet. At the moment this should be a dict with an attribute named regions: regions: [

'eu-west-3', 'sa-east-1',

]

Parameters **config** – The dict containing the configuration used for puppet

CHAPTER

FIFTEEN

PROJECT ASSURANCE

15.1 Assurance

This project has been through an assurance process to ensure the project is:

- valuable to AWS customers
- properly licenced

The same process ensures that there are mechanisms to ensure maintainers are:

- likely able to acceptably support it with regards to being responsive to github issues and pull requests

And finally, at the time of publishing:

- any 3rd party components actually contained in the repo are checked to ensure they are correctly licensed and that we are correctly complying with the open source licenses that apply to those 3rd party components.

15.2 Project Management

15.2.1 Quality Assurance

CICD Process

Unit tests are run on every commit. If unit tests fail a release of the project cannot occur.

The project dependencies are scanned on each commit for known vulnerabilities. If an issue is discovered a release of the project cannot occur.

Review Process

There are regular reviews of the source code where static analysis results and unit test coverage are assessed.

15.2.2 Raising a feature request

Product feature requests drive the majority of changes to this project. If you would like to raise a feature request please raise a Github issue.

15.2.3 Backwards compatibility

All changes to date have been fully backwards compatible. Effort will be made to ensure this where possible.

15.2.4 Design consultation

When there is a significant addition or change to the internal implementation we consult a limited number of users. Users are asked to access the potential impact so that we can understand the impact and the potential value of the change. If you would like to register as such a user please raise a Github issue.

CHAPTER

SIXTEEN

CONTRIBUTING

16.1 General Advice

If you are planning on building a feature please raise a github issue describing the requirement to ensure someone else is not already building the same capability.

When you raise the github issue please explain how you are going to architect the capability. There are some pointers here on design principals but it would be good to get some visibility on what you are planning sooner rather than later.

16.2 Building locally

You can build and run this framework locally. There is a Makefile to make this easier for you. In order to run the Make targets you will need to have python poetry and python 3.7 installed. We recommend using pipx to install poetry.

Running `make help` will display a description of the targets along with a description of what they do.

PYTHON MODULE INDEX

S

INDEX