aws-service-catalog-puppet

Eamonn Faherty

Oct 08, 2019

CONTENTS

CHAPTER

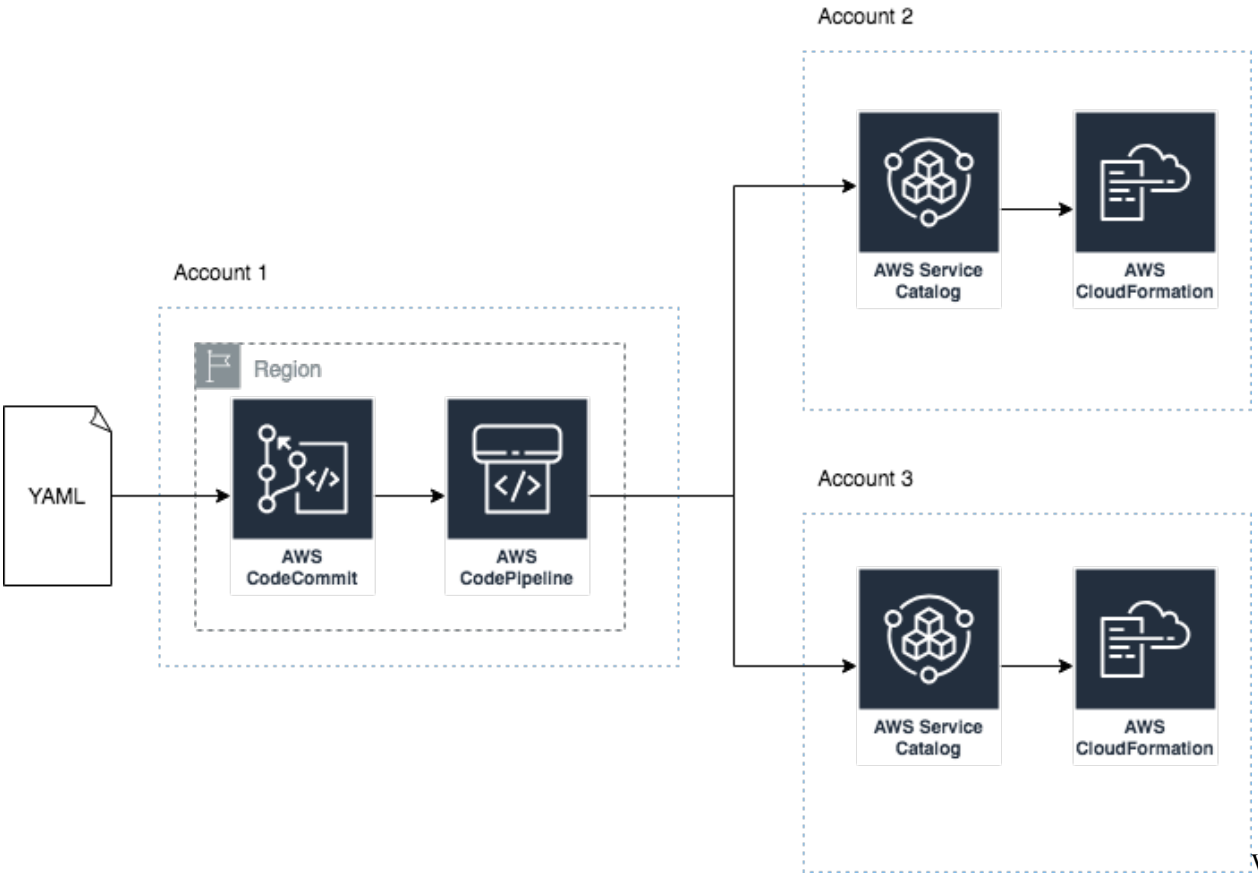
ONE

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 *hub* 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

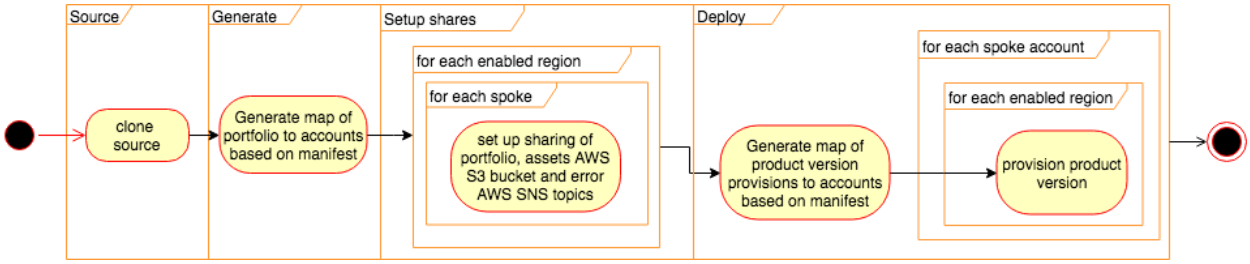
TWO

GETTING UP AND RUNNING

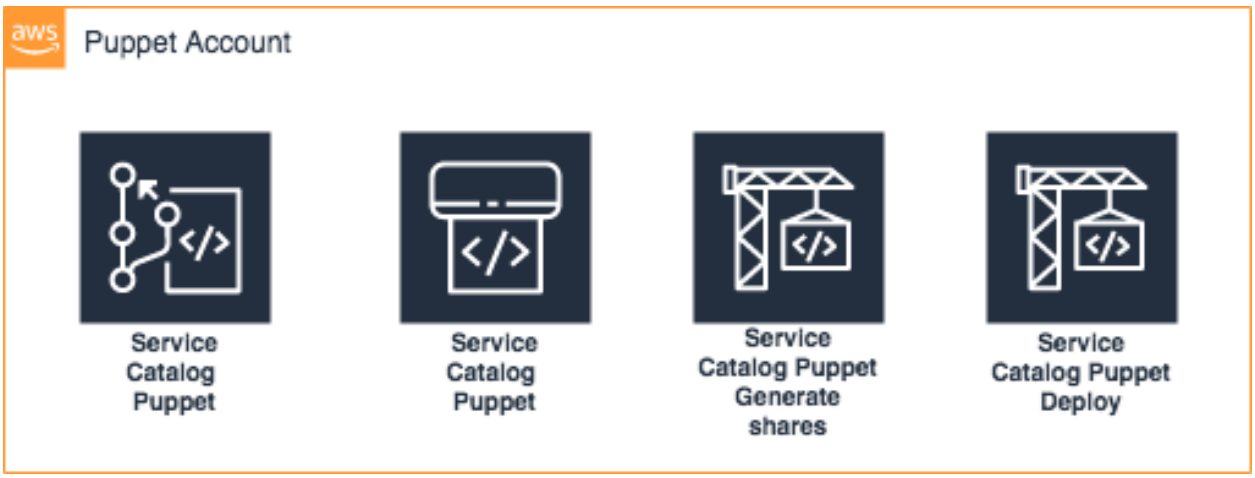
ServiceCatalog-Puppet runs in your AWS Account. In order for you to install it into your account you can use the aws-service-catalog-puppet cli. This is distributed via [PyPi](https://pypi.org/project/aws-service-catalog-puppet/)

2.1 What am I going to install?

ServiceCatalog-Puppet is bootstrapped from your local machine. You install a command line utility that will provision the resources you need into your AWS Account. Once you have completed the bootstrap 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 the tool

This is a python cli built using Python 3.

It is good practice to install Python libraries in isolated environments. You can create the a virtual environment using the following command:

```
virtualenv --python=python3.7 venv
source venv/bin/activate
```

Once you have decided where to install the library you can install the package:

```
pip install aws-service-catalog-puppet
```

This will install the library and all of the dependencies.

2.4 Setting it up

The Puppet will run in your account and needs some configuration. You will need to stand up the puppet and set up the configuration for it to run smoothly.

You will also need to provision an IAM Role within the _spoke_ accounts - those you want to provision products in.

2.4.1 Bootstrap your spokes

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

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

2.4.2 Bootstrap your account

There are two or threes parts to bootstrapping the puppet.

Setting up your global configuration

The first is concerned with setting the global configurations. To do this we use AWS SSM Parameters. To get setup you need to create a configuration file with a list of regions you want to use. You must also specify if you want to collect the CloudFormation events from provisioning actions via SQS.

Here is an example config.yaml:

```
regions: [
  'us-east-2',
  'us-east-1',
  'us-west-1',
  'us-west-2',
  'ap-south-1',
  'ap-northeast-2',
  'ap-southeast-1',
  'ap-southeast-2',
  'ap-northeast-1',
  'ca-central-1',
  'eu-west-1',
  'eu-west-2',
  'eu-west-3',
  'sa-east-1',
]
should_collect_cloudformation_events: false
should_forward_events_to_eventbridge: true
should_forward_failures_to_opscenter: true
```

Note: should_collect_cloudformation_events was added in version 0.33.0

Note: should_forward_events_to_eventbridge was added in version 0.35.0 should_forward_failures_to_opscenter was added in version 0.35.0

Once you have this file you need to upload the config:

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

If you make changes to this you will need to run upload-config and bootstrap commands again for the changes to occur.

Once that has completed you are ready to bring up the rest of the puppet.

Setting to to use AWS Organizations

The second part to bootstrapping is optional. If you would like to use AWS Organizations features in your manifest file you will need to set which IAM Role should be used to perform these actions.

To create the correct role in your organization master export your credentials or change profile and run the following:

```
servicecatalog-puppet bootstrap-org-master <ACCOUNT_ID_OF_YOUR_PUPPET>
```

This command will provision a role the account you specified and output the ARN of the role.

Once you have the ARN or you know the ARN you want to use you can configure the framework to use it. Export the credentials for your puppet account or change your profile so you are using your puppet account and run the following command:

```
servicecatalog-puppet set-org-iam-role-arn <THE_ARN_YOU_WANT_TO_USE>
```

Once you have run that command you are ready for the final stage.

Configuring your puppet

When you bootstrap your account you can choose whether to have a manual approval step in your deployment pipeline.

If you choose to enable manual approvals an AWS SNS Topic with the ARN `arn:aws:sns:${AWS::Region}:${AWS::AccountId}:service-catalog-puppet-dry-run-approvals` will be created to notify you when approvals are required.

To start the bootstrap process you must run the following in your master account:

```
servicecatalog-puppet bootstrap
```

If you want to enable manual approvals you must bootstrap using the following command:

```
servicecatalog-puppet bootstrap --with-manual-approvals
```

2.4.3 Setup your puppet

Clone the configuration repo and configure your factory by editing the `manifest.yaml` file:

```
git clone --config 'credential.helper=!aws codecommit credential-helper $@' --config
↪'credential.UseHttpPath=true' https://git-codecommit.eu-west-1.amazonaws.com/v1/
↪repos/ServiceCatalogPuppet
servicecatalog-puppet seed simple ServiceCatalogPuppet
cd ServiceCatalogPuppet
vim manifest.yaml
git add .
git commit -am "initial add"
git push
```

Wait for pipeline to complete and you have a working puppet.

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.

3.1.1 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.

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:
      RoleName:
```

(continues on next page)

(continued from previous page)

```
        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. 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
```

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:
    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|
+-----------------------+-----------------------------+-----------------------------
↪-----------------+
```

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>'
    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>'
    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
```

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:
    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
```

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`:

```
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
    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:

```
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
          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

CHAPTER

FOUR

SHARING A PORTFOLIO

4.1 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 copy 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.

4.2 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
    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)

4.2.1 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.

4.2.2 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:
```

(continues on next page)

(continued from previous page)

```
        - 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
```

4.3 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.

4.4 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

FIVE

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

SIX

UPGRADING

Firstly, verify which version you have installed already:

```
servicecatalog-puppet version
```

If this errors, check you have activated your virtualenv.

Then you are ready to install the version you want:

```
pip install aws-service-catalog-puppet==<version>
```

If you want to upgrade to the latest you can run:

```
pip install --upgrade aws-service-catalog-puppet
```

Once you have completed the upgrade you will have to bootstrap your install again:

```
servicecatalog-puppet bootstrap
```

And finally, you can verify the upgrade has worked by running version again:

```
servicecatalog-puppet version
```

CHAPTER

SEVEN

FREQUENTLY ASKED QUESTIONS (FAQ)

7.1 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.

7.2 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
```

CHAPTER

EIGHT

USING THE CLI

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

8.1 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"
    }
)
```

8.2 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: '753572411233'
default_region: eu-west-1
name: '753572411233'
regions_enabled:
```

(continues on next page)

(continued from previous page)

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

8.3 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.

8.4 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: '753572411233'
deploy_to:
  tags:
    - regions: default_region
      tag: type:prod
```

8.5 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>
```

8.6 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.

8.7 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.

8.8 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
```

8.9 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
```

8.10 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
```

CHAPTER

NINE

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:

9.1 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*)
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

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.**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

TEN

PROJECT ASSURANCE

10.1 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.

10.2 Project Management

10.2.1 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.

10.2.2 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.

10.2.3 Backwards compatibility

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

10.2.4 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.

PYTHON MODULE INDEX

S

INDEX