DATA IS POTENTIAL

Demo: Provisioner mocks for CORTX (POC)

Provisioner

21/18/21

SEAGATE

Why?

Why we need mocks?

- **General thoughts**
  - Mocking comes from unit level testing and proviides an ability to:
    - Break relations with most of dependencies and focus on one localized part of the logic only
    - Make verification much (resource) cheaper and faster
  - Provisioner mostly operates on integration levels:
    - SaltStack scripts and states require salt tool and services that we can't mock
  - But the idea worth to adapt for Provisioner integration cases as well to get the similar benefits

Why we need mocks?

- **General thoughts**
  - Mocking comes from unit level testing and proviides an ability to:
    - Break relations with most of dependencies and focus on one localized part of the logic only
    - Make verification much (resource) cheaper and faster
  - Provisioner mostly operates on integration levels:
    - SaltStack scripts and states require salt tool and services that we can't mock
  - But the idea worth to adapt for Provisioner integration cases as well to get the similar benefits
- **Current Provisioner needs:**
  - **Lightweight way of verification for Provisioner's own logic of mini API support**
    - No real packages download, installation (fast, no non-provisioner issues, less deals with pre-requisistes)
    - No real provisioner mini integration (fast, no non-provisioner issues)
    - Less system constraints (e.g. can test that in docker locally on a dev system)

Why we need mocks?

- **General thoughts**
  - Mocking comes from unit level testing and proviides an ability to:
    - Break relations with most of dependencies and focus on one localized part of the logic only
    - Make verification much (resource) cheaper and faster
  - Provisioner mostly operates on integration levels:
    - SaltStack scripts and states require salt tool and services that we can't mock
  - But the idea worth to adapt for Provisioner integration cases as well to get the similar benefits
- **Current Provisioner needs:**
  - **Lightweight way of verification for Provisioner's own logic of mini API support**
    - No real packages download, installation (fast, no non-provisioner issues, less deals with pre-requisistes)
    - No real provisioner mini integration (fast, no non-provisioner issues)
    - Less system constraints (e.g. can test that in docker locally on a dev system)
  - **Locally available logic of release and upgrade bundles build automation**
    - No need to ask RE (Jenkins) to build some for us
    - Can do that locally

What do we have now?

CORTX SW mocks

- Only provisioner mini API inside (setup.yaml) for now
- A template spec

Build bundles tool

- **The tool - buildbundle.sh**
  - Builds mock rpm packages for CORTX
  - Prepares a proper directory structure for a bundle type
    - flat yum repository for cortx OR single repo deploy bundle OR upgrade bundle
  - (optionally) Packs it into ISO
- **Location**
  - Repo: srv/components/misc_pkgs/mocks/cortx/files/scripts/buildbundle.sh
  - Installation path: /opt/seagate/cortx/provisioner/srv/components/misc_pkgs/mocks/cortx/files/scripts/buildbundle.sh
- **More docs**

SaltStack state

- **components.misc_pkgs.mocks.cortx**
    1. Calls build script to build CORTX repository with mocks
    2. Setups a repository
    3. Installs the packages
- **More docs**

Show cases

Present build bundles script

1. **Build simple CORTX repository**

   - bash /opt/seagate/cortx/provisioner/srv/components/misc_pkgs/mocks/cortx/files/scripts/buildbundle.sh -t deploy-cortx -o /tmp/deploy-cortx-repo -r 2.1.0 --gen-iso
   - Verify:
     - ls -1 -d /tmp/deploy-cortx-repo*
     - tree /tmp/deploy-cortx-repo

2. **Build deploy bundle (single repo for a deployment)**

   - bash /opt/seagate/cortx/provisioner/srv/components/misc_pkgs/mocks/cortx/files/scripts/buildbundle.sh -t deploy-single -o /tmp/deploy-single-repo -r 2.1.0 --gen-iso

3. **Build upgrade bundle**

   - bash /opt/seagate/cortx/provisioner/srv/components/misc_pkgs/mocks/cortx/files/scripts/buildbundle.sh -t upgrade -o /tmp/upgrade-bundle -r 2.1.0 --gen-iso

Mock an environment with CORTX SW mocks

1. **Build and setup CORTX mocks**
   - salt "srvnode-1" state.apply components.misc_pkgs.mocks.cortx
   - Verify:
     - yum list installed | grep cortx
     - repoquery --list --installed cortx-motr
     - tree -I '*provisioner*' /opt/seagate/cortx
     - cat /opt/seagate/cortx/motr/conf/setup.yaml
     - less `which mock`
2. **Show we can track provisioner mini calls**
   - salt "srvnode-1" state.apply components.s3server.config
   - Verify
     - cat /tmp/mock.log | grep Stage
   - salt "srvnode-1" state.apply components.csm.backup
   - Verify

Future enhancement

What do we need more?

- **Provisioner roadmap includes many directions**
  - Deployment and new workflow support
  - Scale-out deployment
  - Provisioner mini support
  - SW Upgrade
  - …
- **Common stop factors for development**
  - Resources:
    - even for a minor change we usually need to have at least SSC virtual machines
    - It would be worse in future as we scale
  - Time:
    - full deployment implies setup of many SW that are usually not related to a change

How can mocks help here?

- (general idea) For a test case understand the environment and:
    - Mock interfaces which we want to control
    - Do not mock things that we want to test

Thank you