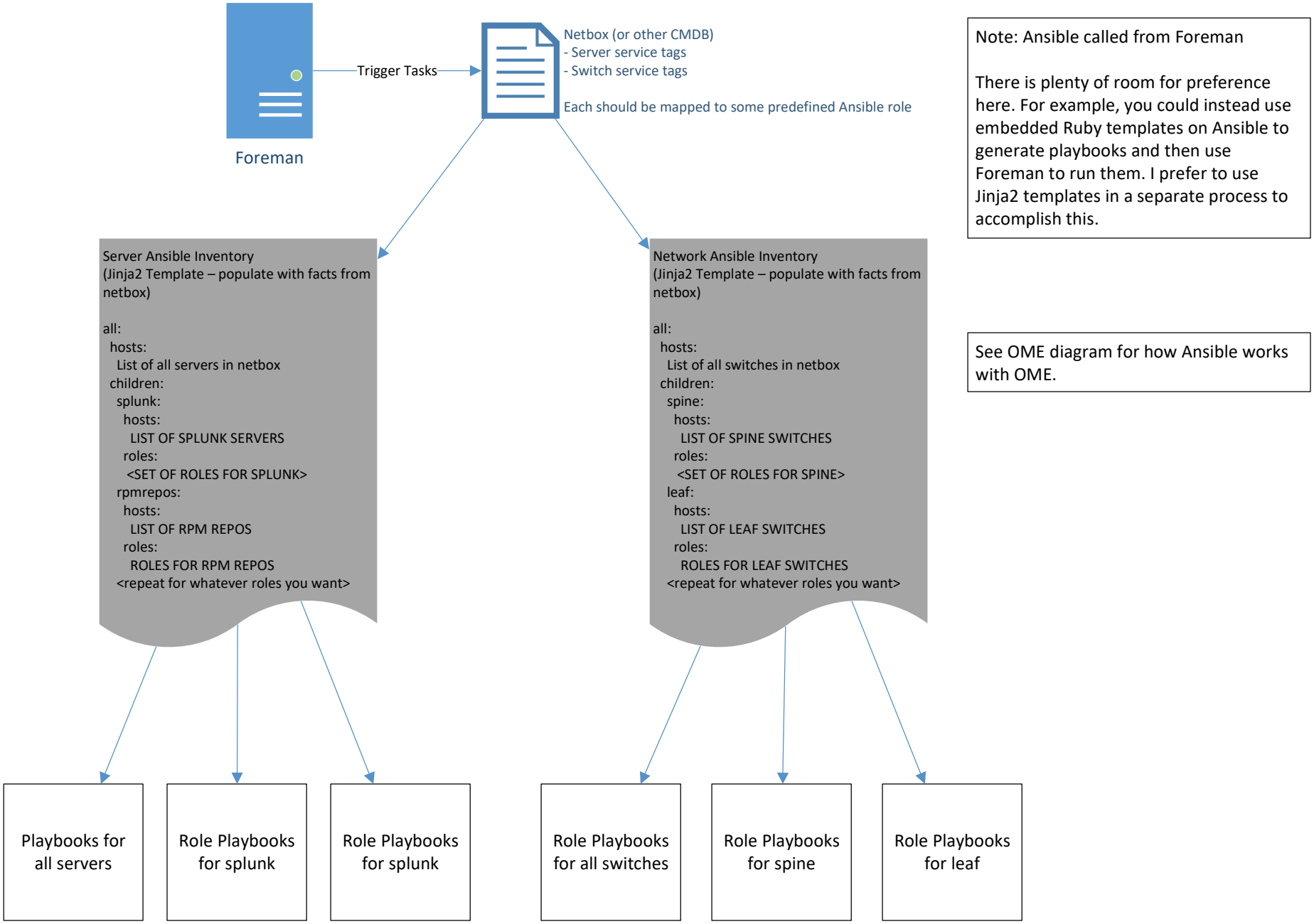
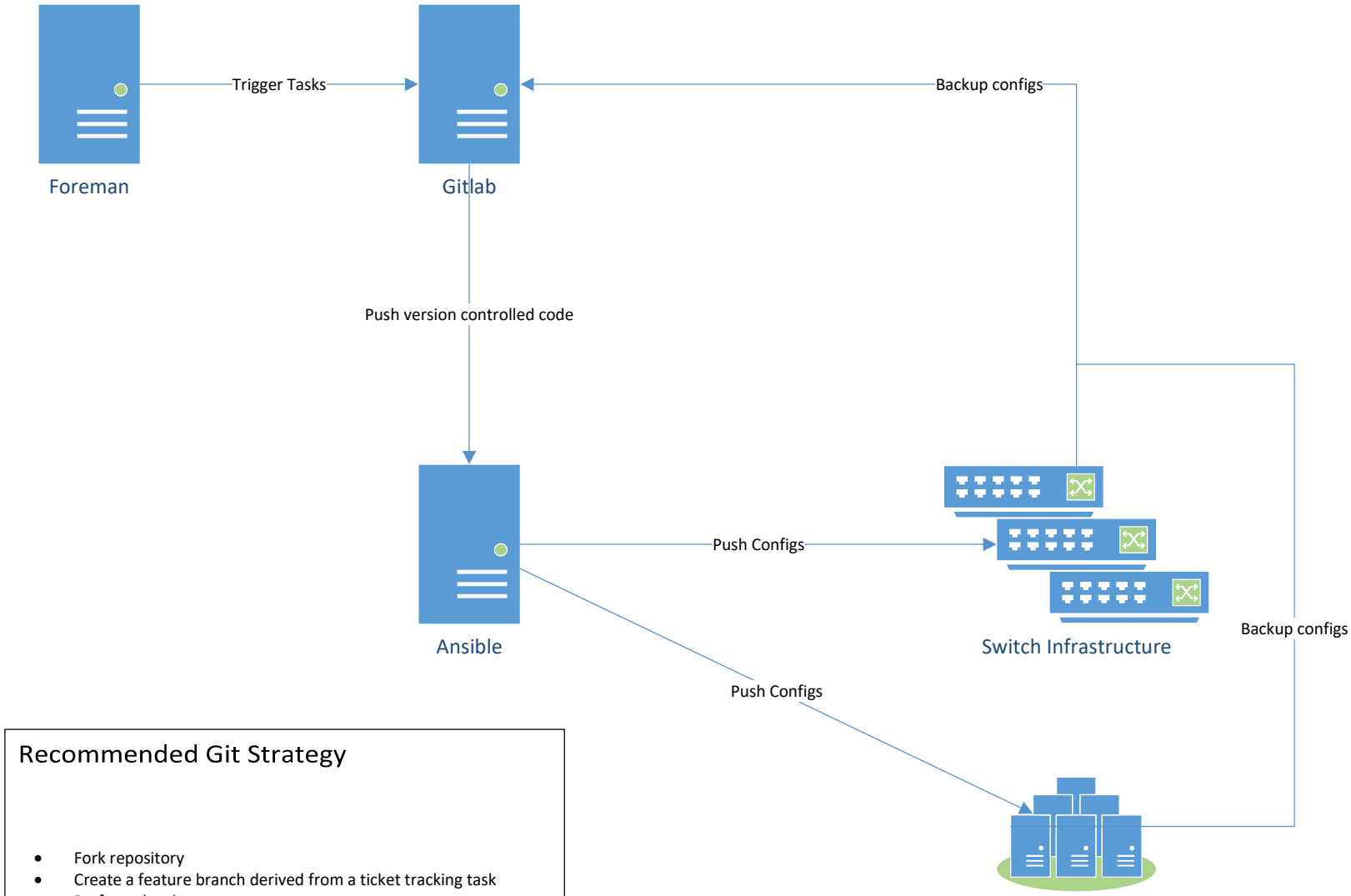


Note: Ansible called from Foreman

There is plenty of room for preference here. For example, you could instead use embedded Ruby templates on Ansible to generate playbooks and then use Foreman to run them. I prefer to use Jinja2 templates in a separate process to accomplish this.

See OME diagram for how Ansible works with OME.



Recommended Git Strategy

- Fork repository
- Create a feature branch derived from a ticket tracking task
- Perform development
- Squash all dev commits into a single commit with a descriptive commit message (use more than one commit when it logically makes sense to do so)
- Rebase feature branch onto current main branch (this may require a force push)
- Pull request feature branch with single commit
- Review pull request and accept into main branch

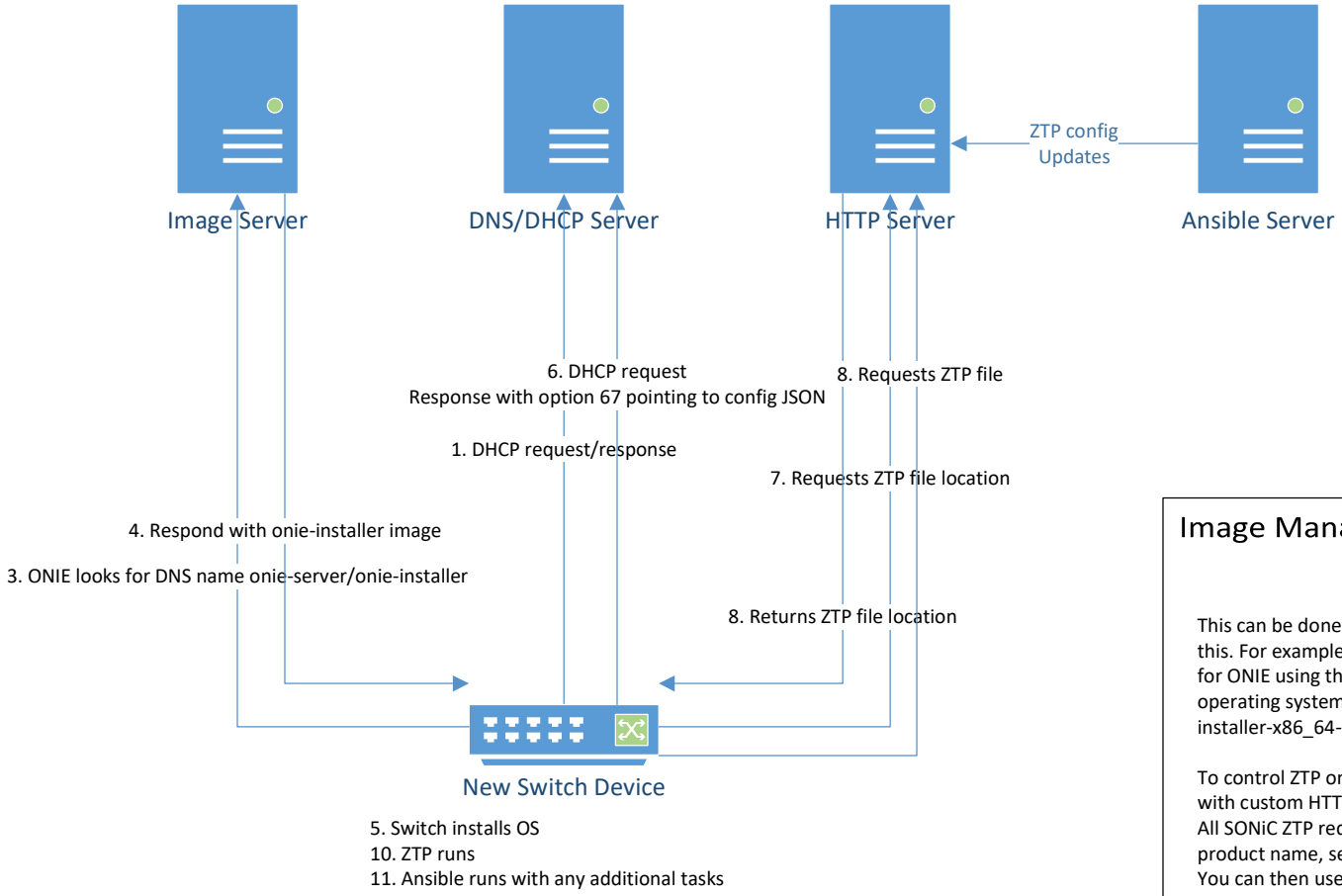


Image Management Strategy

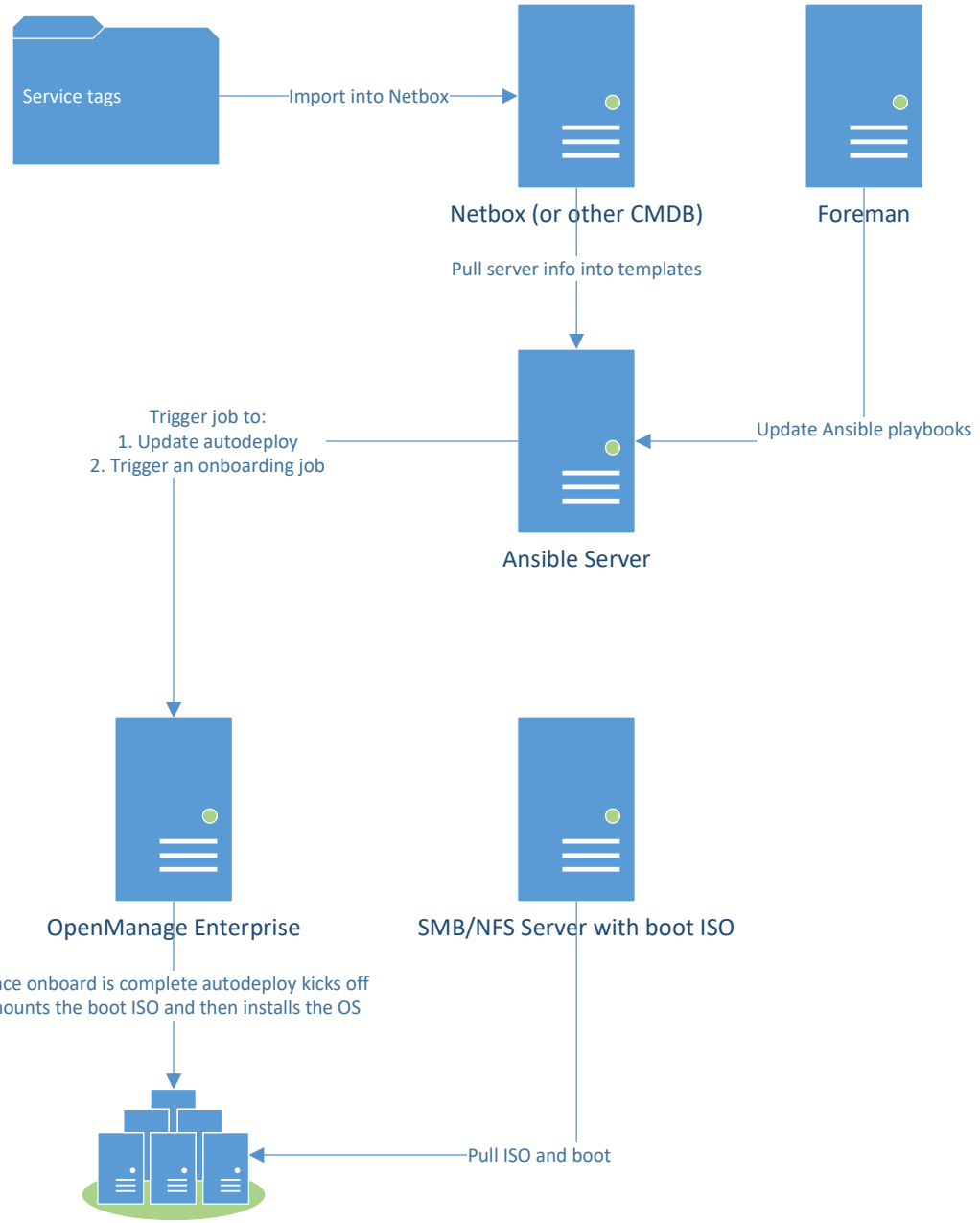
This can be done a number of ways but I generally use Ansible for this. For example, one strategy is to use Ansible to create soft links for ONIE using the switch's model number if you have different operating systems you want to install. Ex: http://onie-server/onie-installer-x86_64-dellemc_z9264f_c3538-r0.

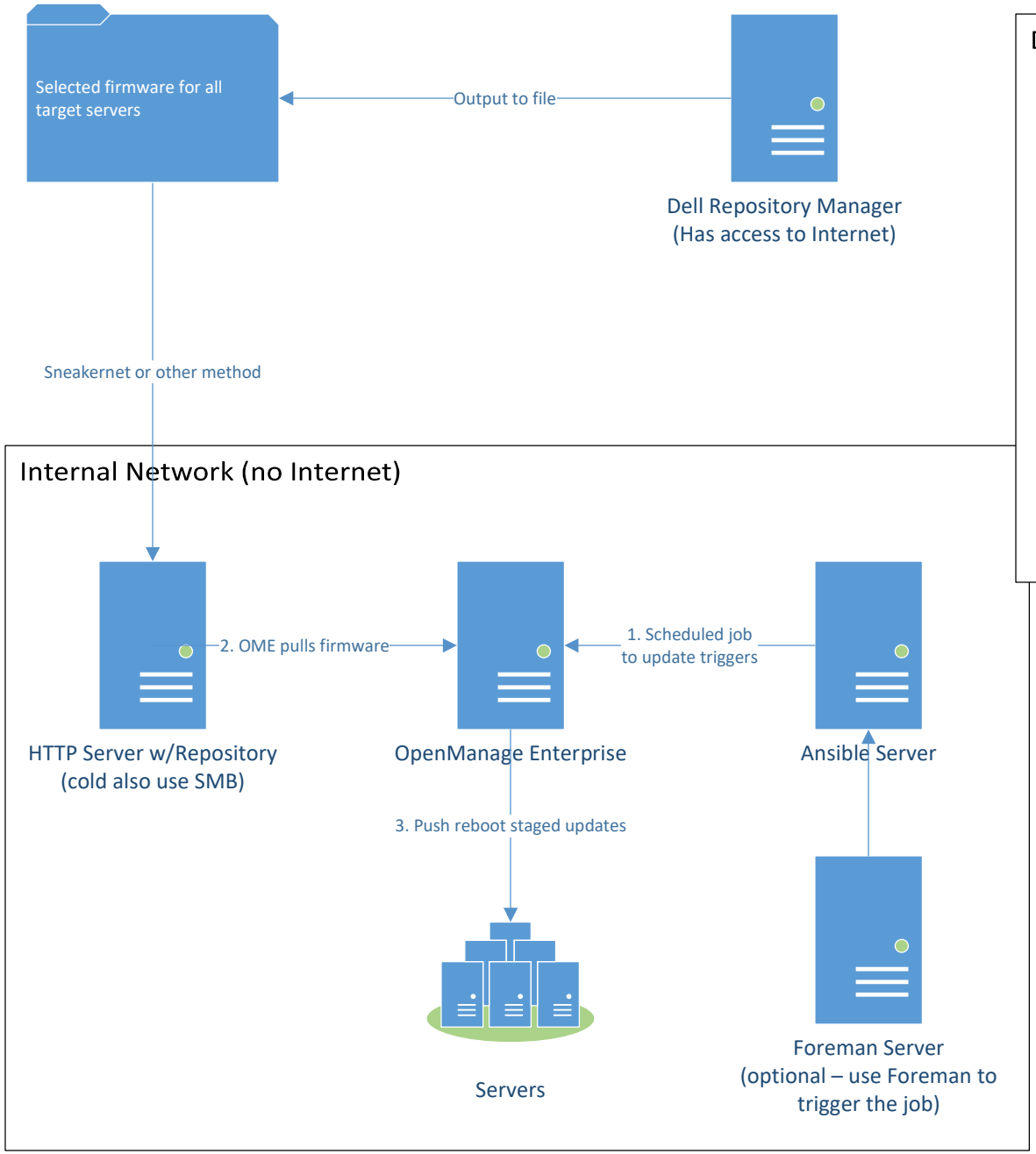
To control ZTP on a per switch basis it is best to use dynamic content with custom HTTP headers. Alternatively you can use dynamic URL. All SONiC ZTP requests include additional HTTP headers including product name, serial number, base mac address, and SONiC version. You can then use this information server side to select a specific ZTP config to send.

Details on workflow

Dell OpenManage Enterprise has a very useful autodeploy feature. As soon as a server is onboarded into OME it will have the autodeploy configuration applied to it which can include applying all desired BIOS templates and then mounting and booting from some target ISO after which further Ansible roles can run.

Alternatively you could use Foreman for everything however keep in mind that the majority of Foreman functionality would require rewrites when BMO is introduced.



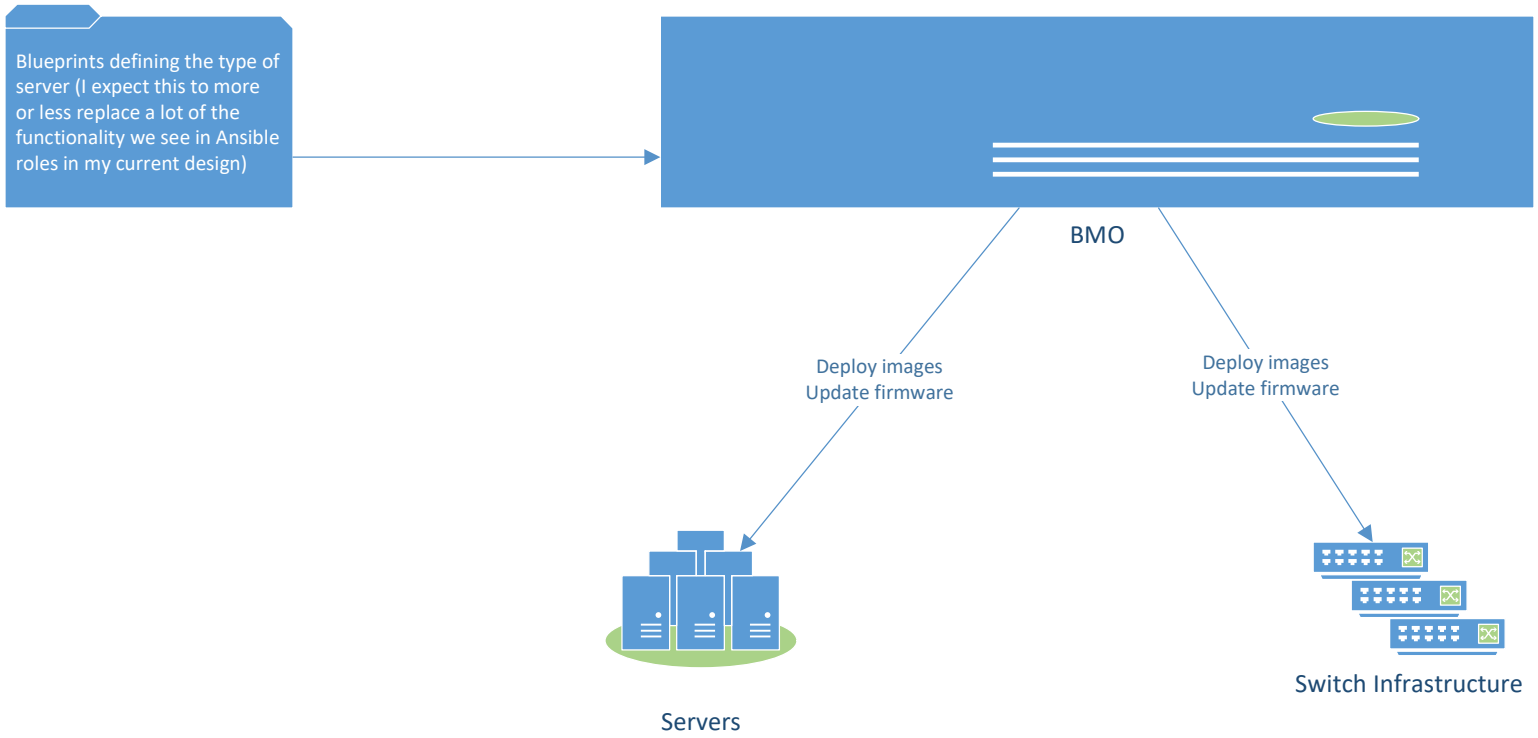


Details on workflow

Dell repository manager (DRM) provides a UI which allows you to select which types of servers you have in your inventory. After you select the desired servers for a given repository you download all firmware from Dell's online catalog. After DRM downloads the firmware you can then export the repository to a target location. The repository is an XML file defining the firmware available along with an organized file structure.

If the target is an offline environment you can burn this to a CD and move it to a target webserver in the offline environment. From there, OpenManage Enterprise can then point at the web server and pull from it to update the machines.

For automation you can use Dell's OME Ansible Modules https://www.dell.com/support/kbdoc/en-us/000177308/dell-emc-openmanage-ansible-modules.

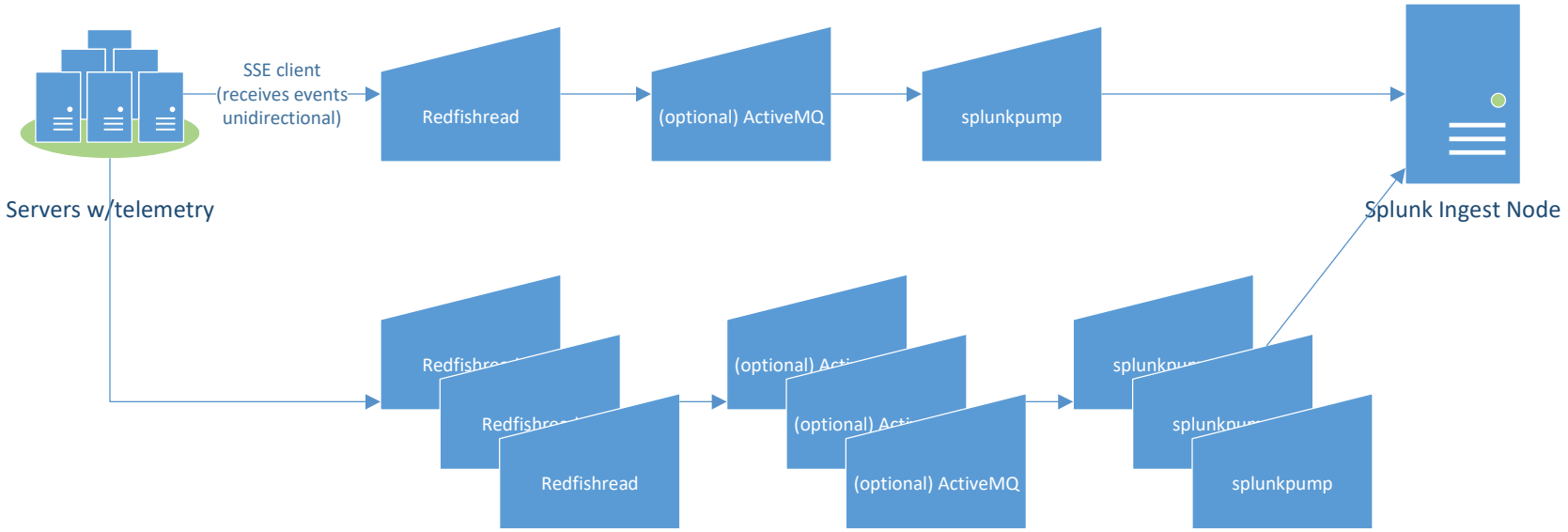


Details

I expect that BMO will more or less completely replace Foreman moving forward which is one of the reasons I would shy away from it in this initial design. I have also deliberately not put much in the way of visualization because I expect BMO to take care of all that. It is going to be the thing that gives you that “central pane of glass”.

A few things off the cuff to expect:

- Firmware management would migrate from OME to BMO
- Write API calls exactly like you would for Foreman for all the same things
- Move integration for autodeploy from OME or other suite to BMO
- Rebuild Ansible to be called from BMO
- Create alerts in BMO



Details

1. Begin by running the appropriate docker compose file. This will start a number of containers detailed below
2. Runs the container telemetry-receiver. This is going to set up all the services required to get the data from iDRAC and into the messaging queue.
    1. Runs idrac-telemetry-receiver.sh inside the container
        1. There are three ways to input configuration variables into the setup and idrac-telemetry-receiver.sh provides provisions for all three:
            1. ConfigUI which provides a simple user interface for controlling config variables
            2. DBdiscauth allows users to configure variables via mysql
            3. simpledisc/simpleauth have file based source identification through the file config.ini
    2. idrac-telemetry-receiver.sh then runs all required go files.
        1. dbdiscauth.go - Runs a mysql DB where users can control the pipeline's configuration variables
        2. configui.go - Runs a lightweight GUI on port 8082 (by default) which allows you to change the configuration parameters of the system (TODO - where does this write to?)
        3. redfishread.go - Sets up an SSE event listener which receives events from the iDRAC and also creates the queue in ActiveMQ. The topic name is databus. This is defined in databus.go by the constant `CommandQueue`
        4. (optional) simpleauth.go/simpledisc.go - Allows the user to input config variables through the config.ini file
3. Runs mysqldb container. mysql provides a mechanism for persisting user settings via a permanent volume mount
4. Runs activemq container - redfishread will pass events from itself to activemq
5. Runs desired database container (elastic/influxdb/prometheus/etc)
6. All networking between the various containers is accomplished with a docker backend network