Helix Management System (HMS) Tight Ship Management

Perforce Professional Services

Version v2023.1, 2023-09-15

Table of Contents

Introduction

The Helix Management System (HMS) has many potential features if fully deployed. The following discusses some aspects of HMS:

- Bastion Host to Access to Perforce Helix servers
- Tight Ship Management
- Monitoring and Alerting
- Host Configuration Management

Each are discussed in this document.

Bastion Host to Access to Perforce Helix servers

The HMS server machine, typically `p4hms.YourDevDomain.net` (where `YourDevDomain.net` is a domain reserved for internal services), is effectively a bastion host for Perforce Helix servers, including the primary Helix Core commit server, any Helix replicas or edge servers, and Helix Broker and Proxy server machines. When logged in as the OS user perforce to the `p4hms`, all other hosts can be accessed without further authentication, as secure and trusted paths have been established with SSH keys.

Tight Ship Management

What is Tight Ship Management?

Tight Ship management entails using version control so that changes to certain files that affect the operation of Perforce Helix Core software are versioned. A tiny p4d service, `p4d_hms`, separate from your other p4d services, is used for Tight Ship Management. The Server Deployment Package (SDP) scripts are versioned on various server machines with HMS, and thus Tight Ship Management is sometimes referred to as SDP Fleet Management.

On each managed server machine, following are versioned:

- The `/p4/common` directory tree
- Any `/p4/N/bin` directory trees (where `N` is any SDP instance; there can be more than on on any given host)
- The crontabs for the perforce OS user

All Perforce Helix server machines, including those running Helix Core Server (p4d), Helix Broker (p4broker) and/or Helix Proxy (p4p) services, are managed using Tight Ship Management.

This means that changes to any files under the noted directories require interaction with version control, i.e. submitting changes made.

A script running in a crontab checks for Tight Ship compliance. The script is: `sdp_sync.sh`. It is is governed by a config file that determines what hosts are managed, `sdp_hosts.cfg`. The script should

not require local updates. The config file should be updated if the Perforce Helix topology changes, e.g. if a proxy is added or decommissioned.

A goal with Tight Ship management is to achieve and maintain Tight Ship compliance. That essentially means that the `sdp_sync.sh` script reports all is well, which means:

- Each managed directory on all machines are in sync with what's in version control, with any discrepancies reported.
- No files are checked out, with any discrepancies reported.
- The live crontabs on each machine match the related versioned files.
- The versioned crontabs are stored in `/p4/common/etc/cron.d`, and are named `crontab.perforce.short_hostname`, e.g. `crontab.perforce.p4d-usw2-01`.

The `sdp_sync.sh` script will email a compliance report daily.

How to work in a Tight Ship environment

Setup Personal P4CONFIG files

There are two ways to work in the Tight Ship environment: You can use the default perforce P4USER user account, or your own personal P4USER account. For accountability, you should use your personal account whenever possible.

On each server machine A P4CONFIG file (see docs for P4CONFIG) exists for using HMS. By default, when you login as perforce, the default shell environment is not setup for managing HMS, but instead for working with other p4d instances. To switch any given terminal shell session into HMS management mode, simply set the P4CONFIG variable.

To act as the perforce OS user (recommended only if you are doing read-only actions), do:

```
export P4CONFIG=/p4/.p4config.SDP
```

Before you act as yourself, make sure you have your own P4CONFIG file, and reference it, which will use a lowercase **sdp** (where the one perforce users uppercase **SDP**), and look like this sample:

```
export P4CONFIG=/p4/.p4config.sdp.jsmith
The file should look like this sample:
P4PORT=ssl:p4hms.YourDevDomain.net:7467
P4USER=jsmith
P4CLIENT=sdp.p4d-usw2-01
P4TICKETS=/p4/hms/.p4tickets
P4TRUST=/p4/hms/.p4trust
P4IGNORE=.p4ignore
```

Once this file exists and the `export P4CONFIG` command is done to reference it, you will need to do a `p4 login` command. Then you can use normal Perforce Helix user commands to open files for and and submit. For example, you might need to update the file

/p4/common/site/hms/config/sdp_hosts.cfg, and could do so like this sample:

```
export P4CONFIG=/p4/.p4config.sdp.jsmith
cd /p4/common/site/hms/config
p4 login
p4 edit sdp_hosts.cfg
vi sdp_hosts.cfg   ## Make your changes.
p4 submit -d "Added new proxy." sdp_host.cfg
```

Example: Updated crontab

In this example, we'll make an update to the crontab on the p4d-usw2-01 server machine. Let's say you have a specific task mind, such as changing the schedule of a cron job.

Generally, you'll start everything by first logging in as perforce@p4hms.p4.YourCorpDomain.net, and then from there ssh to any other server in the environment.

First, SSH to the machine and get setup to use HMS:

```
ssh p4d-usw2-01
export P4CONFIG=/p4/.p4config.sdp.jsmith
```

Then get into the directory where crontabs are, and set a shell variable C for convenience, and check out the file.

```
cd /p4/common/etc/cron.d
C=crontab.$USER.$(hostname -s)
p4 edit $C
```

Next, get the crontab from the machine into the local file, and be sure there are no diffs reported; there shouldn't be at this point.

```
crontab -l > $C
p4 diff
```

Then edit the file, make your changes, diff them, upload the local file back into cron, and submit the versioned file:

```
vi $C # Make your changes
p4 diff
crontab $C
p4 submit -d "Adjusted timing of proxy cache cleaning." $C
```

That's it. This will get the job done and maintain Tight Ship compliance.

Host Configuration Management

Host Configuration Management (HCM) is essentially how to work with HMS when operating as the OS root user rather than perforce. This is used to version specific changes to key system files owned by the root user, and does not apply to files in or under the `/p4` directory tree. When logged in as the root user on the p4hms server machine (and only on this machine), sparse version management is used. That is, only a hand-picked set of individual files are versioned, not entire directory trees. To see the set of files versioned in this way, you can run the command:

```
p4 files //...
```

This will show files under the path `//streams/main/⋯`, which corresponds to the `/` directory, the root of the filesystem. For example, if the file `//stream/main/etc/hosts` is show, that indicates that the `/etc/hosts` file is versioned.

When working as root, there is no live running p4d service to connect to as there is with HMS. Instead, the Helix native DVCS features are used. This behaves much like operating with Git on a personal project – you are working on a private set of files, with no live running service.

As far as interactions go and the p4 commands you use to edit and submit, nothing is different. There are operational differences of Host CM from regular HMS management:

- You don't need to change the P4CONFIG file, as it is already set in the root user's .bashrc and need never change.
- After you make and submit changes to any files, you need to run the command `p4 push` to get them into the HMS server.
- Because there is no live running p4d service when using Helix native DVCS features, you can do versioning as root even if the service is not online. However, the `p4d_hms` service must be running in order to do a `p4 push` from the Host CM server to the HMS server.

Using Host CM has some significant benefits, which are generally the benefits of version control, but applied to specific files on the `p4hms` server machine. Mainly, it is used for:

- Managing P4Prometheus and Alert Manager files, e.g. to make changes to alerting rules.
- Changing other files like `/etc/hosts`, /etc/grafana/grafana.ini`, and `/etc/alertmanager/alertmanager.yml`.
- With versioning, you have the knowledge of knowing what things in key files changed, when, and why (in the form of a change description). This is particularly useful for complex configuration files like `grafana.ini`. And you can easily rollback files to earlier versions if needed. Host Versioning can be augmented by adding a Swarm server to the HMS server, to add notification and optionally code review for changes to any of the files managed with Host CM.

See the file `/root/HostCM/ReadMe.md` on the `p4hms` server machine for additional detail on using Host CM.

Monitoring and Alerting

Prometheus is an open source project used for general purposes systems monitoring. (We actually use Victoria Metrics, a fork of Prometheus). P4Prometheus is the extension to Prometheus provided by Perforce Software that adds knowledge of additional things to monitor on Helix Core server machines. Grafana is a software tool that adds useful visualizations via a web interface to the Prometheus data.

The Prometheus, Prometheus AlertManager, and Grafana services all run on the p4hms server machine, and are managed with `systemctl`, with these service names (illustrated with status commands):

- `systemctl status prometheus`
- `systemctl status grafana-server`
- `systemctl status alertmanager`

The P4Prometheus software is installed and operating on various managed p4d server machines. On each, there is a `p4prometheus` service and a `node_exporter` service, managed with `systemctl`.

Alerting Notification Receivers and Alerting Rules

Alerting rules may need to be adjusted from time to time. The alerting rules are defined in 2 files:

- Notification receivers, i.e. who gets emails, are defined in /etc/alertmanager/alertmanager.yml
- Alerting rules are defined in /etc/prometheus/perforce_rules.yml

Changing Alerting Notification Receivers

To change the receivers, use the following procedure. You will need to be familiar to with the format of the `alertmanager.yml` file; see links:https://github.com/prometheus/alertmanager/blob/main/docs/alertmanager.md[docs here]. Its format is fairly intuitive and you may decide you don't need additional documentation once you see it.

To edit the rules, do like so:

Login as root@p4hms.YourCorpDomain.net

```
cd /etc/alertmanager
p4 edit alertmanager.yml
vi alertmanager.yml   # Make your changes
make
```

The `make` command will perform a validation of your changes. If that indicates success, then restart the service for your changes to take effect, like so:

```
make restart
```

As a shortcut, you can use do only the `make restart`, and skip the separate call to `make` to do only validation. The `make restart` will do the validation first and, if that is successful, restart the service.

Then version the changes in the HostCM server, and push to HMS:

```
p4 submit -d "Your change description" alertmanager.yml
p4 push
```

Changing Alerting Rules

To change the alerting rules, use the following procedure. You will need to be familiar to with the format rules file; see docs here. Its format is fairly intuitive and you may decide you don't need additional documentation once you see it.

To edit the rules, do like so:

Login as root@p4hms.YourCorpDomain.net

```
cd /etc/prometheus
p4 edit perforce_rules.yml
vi perforce_rules.yml   # Make your changes
make
```

The make command will perform a validation of your changes. If that indicates success, then restart the service for your changes to take effect, like so:

```
make restart
```

As a shortcut, you can use do only the `make restart`, and skip the separate call to `make` to do only validation. The `make restart` will do the validation first and, if that is successful, restart the service.

Then version the changes in the HostCM server, and push to HMS:

```
p4 submit -d "Your change description" perforce_rules.yml
p4 push
```