SDP Windows to Linux Migration Guide

Perforce Professional Services

Version v2023.2, 2024-03-27

Table of Contents

DRAFT NOTICE

This document is in DRAFT status and should not be relied on yet. It is a preview of a document to be completed in a future release.

Preface

This guide documents the process for migrating a Helix Core (P4D) service from a Windows server machine to Linux. A migration can be minimally disruptive to users if planned and executed properly. This document informs the planning and execution of a Windows to Linux migration.

Because P4D on Linux can run in the same case-insensitive mode that is familiar to users operating on P4D on Windows, the migration can be nearly seamless to users. After preparation, the eventual cutover is done with a `p4 failover` command in a scheduled maintenance window (or a series of failovers if edge servers or filtered replicas are involved). A failover smoothly transitions the P4D service from one machine (a Windows server machine in this case) to another (the Linux server machine), with no loss of data and minimal disruption.

The preparation typically involves straightforward (though potentially long-running) tasks for the administrator. If the Windows P4D runs with custom triggers or extensions, there will be a degree of complexity depending on how complex those custom triggers and extensions are, and how they are handled. Typical options for handing them include porting or ditching (temporarily or permanently) the custom triggers or extensions. Aside from triggers and extensions, if any additional custom automation operates on the Windows P4D server machine directly, similar handling may be required.

Regardless of the effort and potential complexity handling customization (if any) for admins, the migration can culminate in a nearly seamless transition for users.

Please Give Us Feedback

Perforce welcomes feedback from our users. Please send any suggestions for improving this document to consulting-helix-core@perforce.com.

Chapter 1. Overview

A Windows to Linux Migration has these elements:

- Migration Planning
- Provision New Linux Server machines.
- Install Perforce Helix on Linux.
- Setup Linux Replica server spec on Windows.
- Define adjustments to configurables.
- Pull and verify archives. This may take a long while if there is a lot of data to pull, potentially requiring multiple iterations of the pull/verify process.
- Do a Dry Run of the cutover.
- Port and/or Test Triggers
- Test, Test, Test in the Dry Run environment (which will later be the new Production environment).
- Correct data issues identified in planning and during dry runs.
- Craft a Cutover Procedure.
- Execute the Cutover Procedure.

For purposes of this document, it does not matter if the servers are on-premises ("on-prem") or in a private or public cloud environment such as AWS, Azure, or GCP.

Each of these components is covered in detail in this guide.

Chapter 2. Migration Planning

It is helpful to review the related document SDP Migration and Upgrade Guide. This document discusses a Big Blue Green Cutover (BBGC) style of migration. For a Windows to Linux migration, use a special type of BBGC called a Failover Migration: After preparations are complete, a `p4 failover` completes the migration to Linux (or a series of failovers if edge servers are involved).

The Windows service may or may not be operated using Server Deployment Package (SDP) for Windows. Regardless of whether the Windows service is managed with SDP, the Windows service is largely left alone during the migration. The target environment will always be setup per best practices as implemented with the Linux Server Deployment Package (SDP).

2.1. Plan for User Impact

The migration can be nearly seamless. Typical impacts to users (humans and automation/bots) for a Windows to Linux migration include:

- Users may need to login again after the Linux server machine(s) becomes the live production environment, depending on whether certain configurables like `auth.id` are adjusted during migration.
- If SSL is enabled, users will need to trust the new Linux server machine(s) when they become the live production environment.
- Depending on how user traffic is directed to the Windows server, there may be an impact:
  - If users connect using a P4PORT that includes an an IP address, users will need to change the P4PORT they use.
  - If the failover plan involves changing a DNS name (as opposed to some instantansous method of traffic redirection), there will be delays associated with DNS changes and DNS cache flushing. (The time required for a DNS change varies greatly across organizations, but is typically predicable in any given organization.)
- If the migration is planned to include a change in authentication mechanism, e.g. standard LDAP → SAML/SSO with the Helix Authentication Service (HAS), users will need to adapt to this.
- There will be some amount of downtime for the scheduled cutover. This typically fits in a 2-hour maintenance window, and in best cases can be as little as a few minutes. If the starting p4d version is old (2018.2 or earlier) or ancient (2013.2 or earlier), a longer downtime may be required if the data set is large.

Other than being aware of the above, users to not need to do any special preparation for the cutover. For example, users *do not* need to be concerned about the state of files in their workspaces. Whatever state files in workspaces are in at the time of cutover — checked out to default or numbered pending changelists, shelved or not, etc. — is not affected by the cutover.

2.2. Failover Migration

This document focuses on the failover style strategy. This entails creating a server spec (ServerID) for a standby of the commit server that we'll call `p4d_fs_linux`, that will operate for a time as a

Linux standby replica of the current production Windows commit server. Depending on various factors such as data scale, project priority and complexity, etc. this Linux replica of the Windows commit server may operate for days, weeks or even months before it is ready for the planned and scheduled failover that will promote the Linux standby server to become the new commit server.

This Failover strategy has several benefits:

- Minimum disruption to end users for the cutover.
- Allows for extensive testing of the new Linux server(s) and infrastructure prior to cutover.
- The effect on the original Windows server(s) and infrastructure is minimal.
- Rollback, while hopefully unnecessary, is straightforward.

While planning and preparation will take time and effort, the disruption to end users can be minimal.

If your current method of operating Helix Core on Windows does not produce a regular metadata **checkpoint**, a change is required to get at least some basic form of checkpoint process in place. (If you are not sure what a checkpoint is, see: backup and recovery concepts.)

The Failover strategy requires that the Windows Helix Core P4D service be at version 2019.1 (latest patch) or later. If it is not already at the latest patch available of 2019.1 or a later major version, see [_combining_upgrade_with_migration].

2.3. Custom Triggers and Extensions

The largest single variable affecting effort for a Windows to Linux migration is effort dealing with custom automation, such as triggers or extensions. This can be literally zero effort if there are no custom triggers or extensions (or none that need to survive the migration). If porting and/or testing is required, that becomes a software development and testing project on its own that folds into the larger migration project.

Any custom triggers or extensions will need to be reviewed. Any that can't be discarded will need to evaluated for porting and testing needs.

Triggers written in a native Windows langage such as batch or PowerShell, or operated as compiled .exe files, will need to be ported. Even triggers written in more portably languages such as Python or Perl will need testing and may need adjustment to operate in the Linux environment.

Extensions are written in Lua, the interpreter for which is entirely containdd in the Helix Core p4d binary itself. As such, custom extensions are less likely to require porting. However, they should still be evaluated and/or tested to be sure they have no Windows OS depenencies in their implementation.

Extensions provided by Perforce Software, such as those assoicated with Helix Swarm and the Helix Authentication Service, are inherently cross-platform and do not need to be ported.

If it is acceptable to go without a particular for a time, porting of that trigger could

> be deferred. In that case, the trigger would be disables during the cutover of the Windows to Linux migration, and then re-add the trigger some time after on Linux. Deferring can remove porting of some triggers from the critical path of the migration project.

2.4. Other Custom Automation on Windows Commit Server Machine

Determine whether you have any custom softawre that runs directly on the Windows commit server machine. Custom automation that executes directly on the Windows server machine itself needs to be evaluated for porting and testing needs.

Any automation that merely connects as a client to the Windows p4d server, such as build server farms, need not be considered (other than possibly needing to login or trust p4d again and/or possibly change the P4PORT, as noted in Section 2.1, “Plan for User Impact”). The Linux server speaks the same protocol as Windows.

2.5. Depot Root and Depot Spec Map Fields

This section desribes a change that must be done on the Windows commit server early in the process, before a Linux standby replica can be setup.

> Read this section and be certain you are comfortable with the theory and commands before making any changes to a live production system.

Perforce Helix depot specs have a field named `Map:` that, if used, must be eliminated prior to the deployment of Linux standby servers. This applies to depots that contained archive files to be replicated, including depots of type `local`, `stream`, `spec`, `extension`, `unload`, `tangent`, and `trait`.

To list all of your depots and their types, run this command:

```
p4 -ztag -F "%type% %name%" depots
```

For a Linux server to replicate archives from a Windows commit server, the `server.depot.root` configurable must be set on the Windows commit server, and the `Map:` fields of all depots must be changed to the default value.

Check to see if the `server.depot.root` configurable is set with:

```
p4 configure show server.depot.root
```

If that displays a value, then it is set. Otherwise it is not.

Next, check the `Map:` field of all depots with this command:


```
p4 -ztag -F "%type% %name% %map%" depots
```

Ignore depots of type `remote` or `archive`.

If done carefully, the changes to set `server.depot.root` and clear the `Map:` field of each depot spec can be done non-disruptively on the live running Windows Helix Core service.

The key to making the adjustment to depot spec `Map:` fields and the `server.depot.root` configurable non-disruptive is to understand that the p4d server will use the `Map:` field value *if it is set to anything other than the default*, and otherwise will fall back to the `server.depot.root` configurable as a default location, to find archive files for the given depot. If the value of the `Map:` field of any given depot is `TheDepotName/⋯`, that means the `Map:` field value is not explicitly set, and thus it will fall back to using the `server.depot.root` configurable for that depot. If the value of the `Map:` field is anything other then `TheDepotName/⋯`, then the `Map:` field value will be used to find archive files for that depot.

Your goal in making the change is to make it so that, immediately as/after the depot spec is updated to set the `Map:` field to its default value, the `server.depot.root` will cause the server to look in the exact same physical location for versioned files. Thus, there is no point in time during the procedure where the p4d server cannot locate its archive files, not even a nanosecond.

Before making changes, the singular `server.depot.root` value must be made to work for *all* existing depots. Make the single `server.depot.root` path work without actually moving any versioned files by using Windows directory symlinks. If individual depots are on different drives, put symlinks to all depots in the directory pointed to by the `server.depot.root` configurable so that p4d can find all depot files from that path. You may also find the Map fields use Windows UNC paths or if Windows junctions.

2.5.1. Sample Procedure to Prepare for Archive Replication

You mission is to make it so all depots find there archive files using *only* the `server.depot.root` configurable. Do whatever tricks with symlinks are needed to make it appear to the p4d server that all depot storage directories appear under the directory referenced by the `server.depot.root` configurable.

Victory looks like having the `server.depot.root` point to a single directory, say `S:\P4Depots`, and that `S:\P4Depots` contains either a subdirctory or a symmlink for all other depots.

STEP 1: Review all Depot Specs

Examine the current `server.depot.root` value and the `Map:` field values for depots of all types other than `remote` or `archive`. Start with:

```
p4 configure show server.depot.root
p4 -ztag -F "%type% %name% %map%" depots
```

Get a sense for the any patterns. Do all depots have a non-default `Map:` field set? Do all depots instead have the default `Map:` field value, and thus fall back to the `server.depot.root` setting?

STEP 2: Set server.depot.root

Choose an appropriate value for `server.depot.root`. If the `server.depot.root` configurable was already set, it need not be changed.

STEP 3: Create Directory Symlinks

In the `server.depot.root` directory

EDITME: Add an example illustrating multiple depots with a mix of different `Map:` field values (and some unset), and using `MKDIR /D` in the `server.depot.root` dir to create symlinks, EDITME: Add reference to clear_depot_spec_Map_fields.sh.

2.5.2. Remote Depots

Depots of type `remote`, which are references to paths in entirely separate Helix Core data sets, are unaffected by Windows to Linux migrations.

2.5.3. Archive Depots

Special planning is required if there are any depots of type `archive` containing digital assets archived with `p4 archive`. Archive depots were intended to work removable storage, and removable storage on Windows server machines will not likely be compatible with Linux server machines due to filesystem differences. The most basic strategy is to `p4 restore` all archive revisions and then re-archive them after the migration. This may require provisioning a different type of removable storage device. Restoring before migration is the recommended approach unless it is not pragmatic for some reason.

2.6. The journalPrefix

The Windows commit server must have the `journalPrefix` value set in order to set up the Linux replica. It can be set to any value that works to enable the p4d service to find numbered journals that have already been rotated. However, if no value is set at all, a value must be set before the Linux replica can be setup.

In some cases, the `journalPrefix` is undefined as a configurable, but a `journalPrefix` is provided as an argument to custom scripts that create checkpoints or rotate journals. In these cases, the `journalPrefix` for the Windows commit server should be set to the same used in custom checkpoint scripts.

In other cases, the `journalPrefix` is undefined, and so numbered/rotated journals (if there are any) appear in the default location in the P4ROOT directory. In these cases, an appropriate journalPrefix value should be set immediately, with a value set so that journal land in an appropriate directory.

A sample command to set the journalPrefix is:


```
p4 configure set journalPrefix=C:\P4Data\checkpoints\p4_1
```

2.7. Find Incompatible Configuration Settings

Using the `p4 configure` command to interact with `db.config` is a good way, and in many cases the only way, to set various configuration items with a Helix Core server. However, there are certain settings that must not be defined with `p4 configure`, as they conflict with settings the SDP defines with shell environment variables on Linux.

Review the output of the command `p4 configure show allservers` and see if any of the following are set for the `any` config (the global defaults):

- `P4JOURNAL`
- `P4PORT`
- `P4LOG`
- `P4TICKETS`
- `P4TRUST`

If any of these are set with `p4 configure`, the migration plan will need to deal with unsetting them after first ensuring they are set in some other way on the Windows service. Following is an example of how to replace how P4LOG is set displays in the output of `p4 configure show all servers`. Note that changing this requires a brief service restart to take effect.

2.7.1. Sample Procedure to replace P4LOG configurable

This is an example of to unset a P4LOG setting if it is set in such a way that it shows up with a `p4 configure show allservers` command. The goal is to unset the configurable, but replace it with a Windows service setting so there is no effective change in the value used on Windows.

This sample assumes the Windows service name is `Perforce` and the `p4 configure show allservers` output contained a setting of P4LOG with a Windows path.

First, set the `P4LOG` setting and associate it with the Windows service name:

```
p4 set -S Perforce P4LOG=L:\p4logs\p4d.log
```

That will set the P4LOG variable so that it is associated with the Windows service named `Perforce`. Once that is done, it can be unset as a configurable, such as in this example:

```
p4d.exe -r E:\PerforceRoot "-cunset P4LOG"
```

Next, stop and then start the Windows service as you normally would.

If the P4LOG setting was set for the `any` config, and there are multiple servers in

> your topology, be sure to set the value for the Windows service on all p4d server machines before unsetting the global default configurable.

2.7.2. Other Windows Paths in Configuration

Also scan the `p4 configure show allservers` output for other settings that contain Windows paths, such as Structured Logs defined to reference a Windows path. Such things will need to be be overridden in the server spec for the Linux replica. For example, if you see:

```
any: serverlog.file.11=E:\PerforceRoot\triggers.csv
```

You'll want to create an override for the Linux replica by doing:

```
p4 configure set p4d_fs_linux#serverlog.file.11=/p4/1/logs/triggers.csv
```

2.8. Uncompressed Journals

Examine how checkpoints and journals are currently taken in the Windows environment (or if they are taken at all).

If journals on the Windows service are compressed, replication will not work. Replication requires uncompressed journals.

One thing to check for is whether scripts call the `p4d -jc` or `p4d -jd` command with the `-z` option (lowercase 'z'). If so, the '-z' should be changed to `-Z` (uppercase 'Z'). The lowerase `-z` compresses both checkpoints and numbered journal files, and thus is not suitable for replication. The uppercase `-Z` compresses the checkpoint file, but not the numbered journal files, and is intended for replication. Other changes to custom scripts that manage checkpoints in the Windows environment may be warranted.

2.9. Set Target P4D Version

Define the desired target P4D version. For illustration in this document, we will assume a target P4D version of 2023.1; 2023.2 or later will also be appropriate.

2.9.1. If Windows P4D is 2019.1+

If the Windows P4D version is 2019.1 or later, there are two viable options to upgrading:

- The Windows infrastructure can be upgraded in place to the target P4D version immediately, before the Linux replica is setup. This will require a service outage for the Windows service early in the project, before the Linux replica is setup. Because the P4D version is 2019.1 or later, upgrades will be relatively straightforward.
- Leave the Windows P4D version as it is. In this case, the plan should account for doing the P4D upgrade in the Linux topology immediately after the failover that promotes the Linux server to

be the commit server, in the same maintenance window (before testing and turning the system over to users).

2.9.2. If Windows P4D is 2013.3 to 2018.2

If the starting P4D version is older than 2019.1 but at least 2013.3, the plan must account for first upgrading the Windows service in place to the target version.

A special upgrade procedure is required for upgrades that go to-or-thru P4D 2019.1.

As a general note, in all cases for P4D upgrades, a single "hop" is done, from the original P4D version directly to the new target P4D version. That applies even if the starting P4D version is an antique 1995.1 and the target version is a modern 2023.2. There is neither need nor value in breaking the upgrade up into multiple steps. Upgrades that go thru certain major versions where architectural changes were made (e.g. 2013.3, 2019.1, etc), additional and specific upgrade procedures will be needed based on the starting and target P4D version.

2.9.3. If Starting P4D is 2013.2 or older

If the starting P4D version is older than 2013.3, a checkpoint replay is required.

Other strategies can be considered that would not require upgrading in place if avoiding an in-place upgrade is a priority. That would entail longer downtime and other complexity. Such options are not explored in detail in this document.

2.10. Avoid Case Sensitivity Conversion

Since this document is about Windows to Linux migrations, the data set will naturally and necessarily be case-insensitive at the start of the project. This document does not discuss case sensitivity change, as it is unnecessary for a Windows to Linux migration. If there is a desire to become case-sensitive (for example, to support Linux clients), we advise deferring that as a separate project to be done after the Windows to Linux migration is complete.

A Windows to Linux migration that preserves the original case-insensitive behavior, as described in this document, is minimally disruptive. A case sensitivity conversion is best to defer until the Windows to Linux migration, for several reasons:

- The conversion to case-sensitive can only be done on Linux.
- Case sensitivity conversion can be disruptive to users and workflows, and may result in data loss (although data that will be lost will be known before the loss).
- Case sensitivity conversion requires significant downtime.
- Case sensitivity conversion requires duplication of 100% of versioned file storage (during development and testing of the case conversion process on your data).
- Case sensitivy conversion may potentially disrupt tooling that interacts with your server.

Generally speaking a case sensitivity conversion is more complex than a Windows to Linux

conversion, sufficiently so that we advise relegating case sensitivity conversion to a separate project from the Windows to Linux migration. The case sensitivity conversion, if done at all, can be started after the Windows to Linux migration is complete. The case sensitivity involves doing neurosurgery on your Helix Core data set using the p4migrate utility.

Further discussion on case sensitivity conversions is outside the scope of this document.

Chapter 3. Helix Core Topology

The complexity of a Windows to Linux migration project is naturally affected by the baseline compelxity of the Helix Core ecosystem operating on Windows.

Is your server a single machine, or are there many server machines? In any case, you'll want to think in terms of a "Big Blue/Green Deploy." Every active Windows server machine in the current production topology (the "Blue" servers), including all replicas, edges, and proxies, will all need equivalent Linux server machines to replace them (the "Green" servers). Replicas are straightforward to handle. Handling edges and/or filtered replicas adds complex complexity to be aware of.

Consider what Perforce Helix server machines and services exist in your Windows topology:

3.1. Helix Proxies

In some cases, Linux proxies will have existed with a Windows commit server all along, as running proxies on Linux is advisable even in a topology with a Windows servers (for some of the same reasons that a Windows to Linux migration is popular, such as much faster native filesystems).

Any Windows should be migrated to Linux as well. However, while strongly discouraged, a Windows p4p (proxy) can remain in place with a Linux p4d server topology (so long as it operates in case-insensitive mode, which we assume in this document).

3.2. Helix Brokers

Helix Brokers should be migrated to Linux as well. However, while strongly discouraged, a Windows p4broker can remain in place with a Linux p4d server topology.

If brokers are configured with any custom software (broker "filter" scripts), porting this software to Linux should be accounted for in planning.

3.3. Helix Core P4D Servers

Every Helix Core topology will have exactly one commit server. If there is only a single server in the topology, it is the commit server. It may have additional p4d servers that extend the topology. Following are types of p4d servers (various typess of replicas) and their implications for a Windows to Linux migration:

3.3.1. Edge Servers

For purposes of a migration, edge servers should be classified into one of two categories:

- Edge servers that must survive the migration with workspaces intact.
- Edge servers that must survive the migration, but can lose all workspaces.
- Edge servers that can be discarded.

For Windows edge servers that must survive with workspaces intact, a microcosm of the plan for the commit server can be applied to the edge server. A Linux standby of the Windows edge server can be configured and failed over to before the failover of the Windows commit server.

If there are no custom triggers, the failover of a Windows edge server to its Linux standby can done far ahead of the failover of the Windows commit to its standby — days, weeks, or even months ahead. However, if there are triggers, that generally means all Windows edges servers that must survive with workspaces intact must be failed over to their Linux standbys in the same maintenacne window that the commit server is failed over to, e.g. minutes or hours before.

For Windows edge servers that must survive but for which workspaces are not needed (such as edges that have no human users but service only automated build farms), those will require Linux server machines, but will be created fresh in the Linux environment, with workspaces discarded during the Cutover Procedure. These will be loaded with a standard checkpoint from the commit server, albeit excluding edge-specific tables like `db.have`.

3.3.2. Filtered Replicas

Filtered replicas that must survive the migration are handled in the same way as edge that needs its workspaces. That is, a Linux standby of the Windows filtered replica is setup and failover over to ahead of the commit server failover.

Unlike edge servers, the filtered forwarding replica can always be failed over to long ahead of the commit server, regardless of whether there are custom triggers (as triggers never fire on replicas).

3.3.3. Unfiltered Replicas

Unfiltered replicas are simply reseeded (loaded with a fresh checkpoint) during the Cutover Procedure.

3.3.4. Standby Servers

Standby servers are simply reseeded (loaded with a fresh checkpoint) during the Cutover Procedure.

3.3.5. Distribution Servers

A Windows to Linux migration has no impact to exsiting servers of type `distribution-server`.

EDITME: Should be true, but test this to confirm.

3.3.6. Helix Swarm

Helix Swarm is essentially a client to the Helix Core server, and as such is largely unaffected by a Windows to Linux migration. It may possibly need to change the configured P4PORT it uses to connect to the commit server, as noted in Section 2.1, “Plan for User Impact”). In the case of Helix Swarm, this would involve update its `config.php` and reloading Swarm’s configuration.

3.3.7. Helix Authentication Service

If the Helix Authentication Service (HAS) has been deployed for the Windows commit server, it can be left in place and will be entirely unaffected by and unaware of the Windows to Linux Migration.

Optionally, the HAS service can be moved onto the Linux commit server machine for easier management.

3.3.8. P4DTG

If the Perforce Defect Tracking Gateway (P4DTG) has deployed for the Windows commit server and operates on a separate server machine, it can be left in place and will be entirely unaffected by and unaware of the Windows to Linux Migration.

If P4DTG operates on Windows, it could be migrated to Linux as well, or left in place. However, while there are many compelling reasons to migrate Helix Core to Linux, there is less of a need to migrate P4DTG if it is stable and operating well. Migrating P4DTG to Linux (e.g. if normalization to all Linux infrastructure is a goal) can be done entirely independetly of, or as part of, the Windows to Linux migration project.

Chapter 4. Preparation

4.1. Define Linux SDP Instance Name

See the definition of Instance in SDP parlance.

Set the intended SDP instance name. For purposes of this document, we'll use the SDP default instance name of 1.

4.2. Define Linux Replica on Windows Commit Server

Reminder, setting a value for `journalPrefix` on the Windows commit server and dealing with the `server.depot.root` must be done before setting up the Linux replica. See:

- Section 2.5, "Depot Root and Depot Spec Map Fields"
- Section 2.6, "The journalPrefix"

Address the above items before moving forward.

On the Windows commit server, create a server spec to represent p4d on Linux. Call it `p4d_fs_linux`. Run this command:

```
p4 server p4d_fs_linux
```

Set these field values:

- `ServerID: p4d_fs_linux`
- `Type: server`
- `Services: forwarding-standby`
- `Description: Linux replica of Windows commit server.`

Next, determine a P4PORT value that can be used from the Linux replica server machine to reference the Windows commit server. Getting this to work may entail opening a firewall rule on the Windows commit server to ensure that the Linux server can reach it on whatever port p4d runs on (often 1666).

```
p4 configure set p4d_fs_linux#P4TARGET=_P4PORT_OF_WINDOWS_SERVER_FROM_LINUX
```

Then define other configurables with commands like these, to be run on your Windows commit server:

```
p4 configure set p4d_fs_linux#db.replication=readonly
p4 configure set p4d_fs_linux#rpl.forward.all=1
p4 configure set p4d_fs_linux#rpl.compress=4
```

```
p4 configure set p4d_fs_linux#server=4
p4 configure set p4d_fs_linux#monitor=2
p4 configure set p4d_fs_linux#serviceUser=svc_p4d_fs_linux
p4 configure set p4d_fs_linux#rpl.journalcopy.location=1
p4 configure set p4d_fs_linux#journalPrefix=/p4/1/checkpoints/p4_1
p4 configure set p4d_fs_linux#server.depot.root=/p4/1/depots
p4 configure set p4d_fs_linux#startup.1="journalcopy -i 0"
p4 configure set p4d_fs_linux#startup.2="pull -i -L 1"
p4 configure set p4d_fs_linux#startup.3="pull -i -u"
p4 configure set p4d_fs_linux#startup.4="pull -i -u"
p4 configure set p4d_fs_linux#startup.5="pull -i -u"
p4 configure set p4d_fs_linux#startup.6="pull -i -u --batch=50"
p4 configure set p4d_fs_linux#startup.7="pull -i -u --batch=50"
p4 configure set p4d_fs_linux#startup.8="pull -i -u --batch=50"
```

Next, create the replication servie user, and set a password for it (generate or think of a password first):

```
p4 --field Type=service user -o svc_p4d_fs_linux | p4 user -f -i
p4 passwd svc_p4d_fs_linux
```

Store the password securely, however you store passwords (e.g. in some kind of password vault).

Next, add the `svc_p4d_fs_linux` user to a group name `ServiceUsers`, and ensure that group has a `Timeout` value set to `unlimited`. Then add this line near the bottom of the Protections table:

```
super group ServiceUsers * //...
```

Once the above metadata change are complete, the Linux replica is defined. The next routine checkpoint to be taken in the Windows environment will have the definition of the Linux replica "baked in", and thus it can be used to seed the Linux replica.

4.3. Provision New Linux Server Machines

EDITME - Add content here.

4.3.1. Select Operating System

As of this writing, the best options are:

- Ubuntu 22.04 (or 20.04)
- RHEL/Rocky Linux 9 (or 8)

4.3.2. Install Helix Core Software on Linux

On the Linux server machines that do not yet have any data, use the Helix Installer, do a Configured

Install.

The Helix Installer is only to be used on truly "green" server machines, those with *absolutely no* Helix Core data on them yet.

Run these commands as `root` on the server machine:

```
mkdir -p /hxdepots/reset
cd /hxdepots/reset
curl -L -s -O
https://swarm.workshop.perforce.com/download/guest/perforce_software/helix-
installer/main/src/reset_sdp.sh
chmod +x reset_sdp.sh
./reset_sdp.sh -C > settings.cfg
```

In `settings.cfg`, change these settings:

- DNS_name_of_master_server=
- P4_PORT=
- Instance=
- Password=
- CaseSensitive=0
- P4USER=
- ServerID=
- ServerType=
- P4BinRel=
- P4APIRel=

Then run the script:

```
./reset_sdp.sh -no_sd -c settings.cfg 2>&1 | tee log.reset_sdp.txt
```

```
su - perforce
p4 set
```

```
cd /p4/common/site
[[ -d config ]] || mkdir config
cd config
```

4.4. Pull Archive Files to Linux

Once the Linux replica are setup, a variety of strategies can be used to transfer archive files.

Plan to execute about 3 iterations of `p4verify.sh` on Linux, to get p4d to pull the archives. The first pass, starting with no archive files, is to start a bulk pull. That could take hours, days or weeks depending on data scale. Also, it may need some nudging, clearing and resetting the replication "pull queue."

The second to fill in gaps, and the 3rd pass should be clean.

Depending on scale of data, you may want to consider using outside-p4d mechanisms for transferring some archives (especially the `.gz` files, `,v` files should be transferred with `p4 pull` ideally).

There are lots of variations on how to get the archives files there. This document focuses primarily on using replication (i.e. replica `startup.N` threads calling `p4 pull` commands) for these reasons:

1. It has an advantage in that, if the Linux p4d writes an archive, it can always find it, even it it's a funky path with Unicode bytes in the path. By contrast, files copied outside p4d may not be found by the Linux p4d if the path to the file (including the base filename and any directory in the path) contain any high-byte, non-ASCII characters.
2. It can be throttled up or down (by configuring more or fewer `startup.N` threads and tuning batching parameters).
3. It is generally safe and non-disruptive to the production Windows environment.
4. It requires no special setup.

The above noted, for initial, first-time bulk pulls of Terabytes of data, a Windows port of rsync might be considered for pulling `.gz` archives files (and only those). It may well pull bulk archives faster than replication. However, rsync entails extra setup effort (not covered in this document), and also has greater risk of impacting the production Windows environment.

A live running rsync daemon is required to run on Linux for the Windows port of rsync to talk to. For Linux to Linux transfers, the rsync utility does not require a live running rsync daemon, but one will be required for this scenario. The daemon service can be crafted with a configuration to land files in a location relative to a depot root directory on Linux that mirrors the path relative to the depot root on Windows.

There are many options here; somehow or other the goal is to get the archive files in place so `p4verify.sh` is happy.

4.5. Define other Metadata Changes for Linux

During the migration to Linux and the Server Deployment Package, a script should be crafted that defines best practice configurables to be set during the cutover, immediately after the failover and upgrade. The Linux server testing should be done with these best practices in place.

The SDP contains a `configure_new_server.sh` that defines best practices for a new server, mostly by running many `p4 configure set` commands to set various configurables. This should not be used exactly as it is because, as the name implies, it is intended for an entirely new server. However, it should be used as a guide to develop a custom script specific to this migration. Typically the procedure starts by creating a script named something like `configure_THIS_server.sh`, initially copied from the stock SDP script `configure_new_server.sh`.

Then the `configure_THIS_server.sh` script is edited in light of the output of `p4 configure show` on the commit server. Generally the editing involves:

- Remove code in the script that creates a `spec` and `unload` depots if those already exist or are not desired.
- Remove setting any configurables that do not need to be set because they already are in the data set.
- Remove setting any configurables that do not need to be set because a locally tuned value is better for the environment than the script default.
- Adjust setting any configurables to desired values.

Ultimately, the goal of developing this script is to capture the results of the process of reviewing and applying best practice configurables to the current data set. The script typically takes a few hours to develop and review, but can be executed in mere seconds during the cutover. This script should be exercised during the dry run and again for the Production Cutover.

4.6. DRY RUN

At least one Dry Run is required to confidently execute a migration. Plan to have at least one.

In the dry run, the `p4 failover` command is NOT used, nor is user traffic directed to the Linux server. Instead, the Linux service is stopped, and the `$P4ROOT/server.id` file is simply hand-edited to be the ServerId the the commit server. Then the service is restarted.

At that point, the Linux commit server will believe itself to be the new commit server, even though users will still be using the Windows server for real work. Then the Linux server can be tested in various ways:

- Test connectivity from all user access points.
- Test connectivity from all server access points, including replicas, proxies, and any integrated systems such as Jenkins, Swarm, P4DTG, etc.
- If there are any `ldap` specs, ensure the targeted LDAP servers can be reached from the Linux server. (This may require firewall adjustments).
- If the migration include a Helix Core upgrade, test the new version.
- If configurables were changed, do any testing warranted by the changes to configurables.

In addition to the value of exercising a procedure that is largely similar to the Production Cutover procedure, the dry run procedure is useful in gathering timing info related to various steps in the process. Endeavour to capture timing of key steps in the process.

After all Dry Runs are declared complete, the Linux server environment can be reset to be a replica of the Windows environment once again, and the archive pull process repeated (which should be much faster this time around, as most archives are in place).

4.7. Craft Cutover Procedure

Craft the Production Cutover procedure with learnings from the dry run(s).

Appendix A: Sample Cutover Procedure

A.1. Sample Migration Scenarion

The following is a sample cutover procedure for a topology with a commit server and an edge server, with custom triggers that have been ported to Linux.

The sample instructions assume the `perforce` OS user on the Linux servers has been setup with the proper shell environment, specifically that the `~/.bashrc` has sourced the `/p4/common/bin/p4_vars` file with the appropriate SDP instance parameter.

The preparation for this sample cutover scenario would have included:

- As set of ported and tested Linux custom triggers (replacing former custom triggers on Windows) deployed on all Linux servers as `/p4/common/site/bin/triggers` folder.
- A triggers table suited for operation on the Linux server after it becomes th commit server.

A.2. One Week Prior to Cutover Procedure

STEP 1: Verify Replication

Verify that replication is healthy on the Linux replica, the Windows edge, and the Linux replica of the Windows edge server.

A.3. One Day Prior to Cutover Procedure

STEP 1: Verify Replication

Verify that replication is healthy on the Linux replica, the Windows edge, and the Linux replica of the Windows edge server.

STEP 2: Checkpoint Linux Replicas

On the Linux standby of the Windows commit server, and separately and in parallel on any Linux standbys of Windows edge servers, request a checkpoint:

```
p4 admin checkpoint -Z
```

Do a `p4 info` first and confirm that the target ServerID is that of the Linux edge server, to avoid taking an unintentional "live checkpoint" of the commit server.

Next, on the Windows commit server, execute a journal rotation:

```
p4 admin journal
```

Once this command has been run, it will trigger the Linux server to start taking a checkpoint. On the Linux server, a checkpoint should immediately appear in the checkpoints directory.

The checkpoints directory is '/p4/*N*/checkpoints' for the standby of the commit server, `/p4/N/checkpoints.ShortServerID

Monitor the checkpoints directory and await the appearance of a *.md5 with the same number as the checkpoint. The existence of the MD5 file indicates the successful completion of the checkpoint process.

Use the `watch` utility and wait until the *.md5 file appears on the Linux standby of the Windows commit:

```
watch -n 5 "cd /p4/1/checkpoints; ls -lrt *.gz *.md5 | tail -5"
```

In parallel, use the `watch` utility and wait until the *.md5 file appears on the Linux standby of the Windows edge:

```
watch -n 5 "cd /p4/1/checkpoints.edge_syd; ls -lrt *.gz *.md5 | tail -5"
```

STEP 3: Replay Checkpoint to offline_db.

On the Linux commit and edge servers, replay their local checkpoint files created in the prior step into to the offline_db. This can be done regardless of whether the local p4d service is replicating or even online at all. The replay to the offline_db is nether affected by nor disruptive to the p4d service. It can be done on the commit and the standby in parallel, though can only be one on each machine only after the checkpoint completes and the local *.md5 file exists on the given machine.

```
nohup recreate_offline_db.sh < /dev/null > /dev/null 2>&1 &
```

Monitor until completion with:

```
tail -f $LOGS/recreate_offline_db.log
```

This preparation of the offline_db allows the Linux service to start operation with daily checkpoints with a reasonably current offline_db. It may a day or so behind by the time the cutover occurs. (If for some reason the cutover is postponed by more than a few days, repeat this procedure of creating and replaying checkpoints on Linux to keep the offline_db reasonably current. Repeating the procedure will replace the offline_db with a more recent checkpoint).

A.4. Cutover Procedure

STEP 1: Verify Replication

Verify that replication is healthy on the Linux replica, the Windows edge, and the Linux replica of

the Windows edge server.

STEP 2: Disabled Scheduled Tasks

On the old Windows commit and edge server machines, disable any Scheduled Tasks related to backups or checkpoints. Also ensure no long-running checkpoint or backup operations are in progress that won't be complete by the time of the intended cutover.

STEP 3: Disable Crontabs

On the new Linux commit and edge server machines, save and then disable all crontabs intended for routine production operation (and they may have been left on during dry runs).

STEP 4: Lockout Users with Protections

STEP 5: Stop Services

STEP 6: Start Services

STEP 7: Rotate Journal

STEP 8: Verify Replication

STEP 9: Failover Edge Server

STEP 10: Apply Metadata Changes for Linux

Apply metadata changes required for operation on Linux and commit server now being on SDP.

STEP 11: Failover Commit Server

STEP 12: Do Sanity Tests

STEP 13: Decide: GO/NO GO

STEP 14: Restore Default Protections

STEP 15: Direct User Traffic to Linux

STEP 16: Enable crontabs

Appendix B: Why Migrate?

Migrations from Windows to Linux have been the single most consistent theme in Perforce Consulting in over two decades, for many reasons. The procedures have evolved over time, with the modern "failover style" replication being the latest in seamless cutover.

EDITME Add some of the many reasons.

Chapter 5. DRAFT NOTICE

This document is in DRAFT status and should not be relied on yet. It is a preview of a document to be completed in a future release.