SDP Migration and Upgrade Guide

Perforce Professional Services

Version v2023.2, 2024-03-18

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 document is useful in when migrating an existing Perforce Helix Core installation an the optimal operating environment from any starting condition. Whether the starting environment is a large enterprise or "garage band" scale, on-prem or on a public or private cloud, or on any operating system, this guide can help get to the optimal deployment environment for Perforce Helix Core.

This document focuses on software rather than hardware aspects of an optimal operating environment. While hardware is not discussed in detail, the migration and upgrade plans described in this document provide an opportunity to test and change out hardware.

This document does not discuss case sensitivity or case-conversions. Those are discussed in the SDP Windows to Linux Migration Guide.

This guide assumes familiarity with Perforce Helix Core and does not duplicate basic information in the Helix Core documentation.

Please Give Us Feedback

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

Chapter 1. Introduction

1.1. Optimal Helix Core Operating Environment

Getting to an optimal operating environment for Perforce Helix Core first requires defining various aspects of optimal. Any definition of optimal should deliver on expectations of being reliable, robust, performant, and secure.

For our purposes in this document, optimal specifically means:

- Physical layer (server machines, storage subsystems, etc.) is as desired.
- Helix Core (P4D) version 2019.1+. The P4D 2013.3 and 2019.1 versions were major architectural overhauls requiring special upgrade procedures. Once the P4D version is 2019.1 or later, future upgrades are standardized.
- SDP version 2020.1+. This is the SDP version from which SDP upgrades are automated.
- All Helix server processes (p4d, p4p, p4broker) are operating on Linux servers, on a Linux major version with plenty of support life left in it. As of this writing, that would be RHEL/Rocky Linux 9, or Ubuntu 22.04. RHEL/Rocky Linux 8 and Ubuntu 20.04 are actively supported as well, but with somewhat less runway available. For EOL dates of various Linux distros, see:
  - RHEL
  - Rocky
  - Ubuntu
  - SuSE
  - EOL dates for multiple distros

In public cloud environments, an optimal Helix Core server be deployed instantly in an optimal environment by using the Enhanced Studio Pack (ESP), an offering by Perforce Software available on Amazon and Azure marketplaces.

1.1.1. Optimal Storage and NFS

This document doesn't focus on hardware or storage components of the definition of optimal for Helix Core. Using NFS isn't part of the optimal definition. At small scale, the cost and complexity introduced by NFS may not be worth the various benefits. However, as the scale of data increases to and above tens of Terabytes, options involving more scalable filesystem solutions like NFS start making sense and may even be effectively required. NFS can be used in on-prem and public cloud environments.

ESP installations do not use NFS, and thus would require adjustment to handle very large Helix Core data sets.

1.2. Motivation

1.2.1. Global Topology Upgrades

This document is written to support projects commonly referred to as a **global topology upgrade**, sometimes part of an even larger **infrastructure modernization** effort. Such upgrades are commonly driven by desires to maintain performance, security, supportability, access to new product features, and in some cases to escape custom aspects of local infrastructure.

If *all* aspects of the current environment are already optimal as defined above, you don't need this document. Instead, use the standard upgrade procedure documented in the Upgrades section of the SDP Guide. So for example, if you it's 2024 and you have SDP 2020.1 and P4D 2019.1, you can upgrade easily in place (unless you also want to change the major version of your operating system, or migrate to a different operating system).

1.2.2. Helix Remote Administration

Another potential additional motivation is to get assistance for managing Helix Core servers, or perhaps to get out of that role entirely (perhaps due to departure of key personnel). If there is interest in turning over the keys to your Perforce Helix servers, consider the Helix Remote Administration program (HRA).

To be eligible for HRA, customers must be on an optimal environment. Signing up for the program commonly entails a process referred to as "HRA Onboarding," which essentially means doing a Migration Style Upgrade to an optimal environment, as outlined in this document.

1.3. Migration Style Upgrades

This document focuses on the Migration-Style Upgrade strategy, as opposed to in situ (in place) upgrades. In situ upgrades are preferred when your deployment environment is already optimal, as defined above in Section 1.1, "Optimal Helix Core Operating Environment". If *all* of the aspects are currently in the desired state, you don't need this document. Instead, use the standard upgrade procedure documented in the Upgrades section of the SDP Guide.

If the hardware, operating system, or P4D/SDP versions are not as desired, this guide is for you. A Migration-Style Upgrade is great when you need to make a *big change* in the least disruptive way possible. A key characteristic of a Migration-Style Upgrade is that your original server environment is largely left alone, with little or no change.

Typically the original environment remains available for some time after the upgrade, with the general idea that the old environment can eventually be decomissioned, archived, and/or simply deleted.

1.4. Big Blue Green Cutover

In a Migration-Style Upgrade, a new set of server machines and supporting infrastructure (such as storage) is deployed. This set of servers is referred to as the Green servers or infrastructure. The current, live production equipment is referred to as the Blue servers or infrastructure.

In preparation for an Production Cutover, Helix Core data is brought into the Green environment.

This is usually non-disruptive to users, who are operating on the Blue (current live production) environment.

The Green Environment operates for a time as a test environment, allowing opportunity to test various aspects of the new infrastructure before it becomes the production infrastructure. Depending on risks and needs, testing can be cursory or extensive, lasting days or months.

If your current method of operating Helix Core does not produce a regular checkpoint, a change to the Blue environment will be required to get at least some basic form of checkpoint process in place. This may involve disruptive operations that might need to be scheduled in a maintenance window before the Green environment can be setup initially.

A Big Blue Green Cutover (BBGC) is planned with the eventual goal of cutting over from the entire Blue infrastructure to the Green infrastructure in one single maintenance window. The *Big* in Big Blue Green Cutover indicates that a phased approach cutover is not an option.

In some types of projects, using a phased approach to migrations can mitigate risk. However, in this type of project, a phased cutover actually introduces more risk and complexity, because it requires operating bi-directionally across the Blue and Green infrastructures, which creates its own set of problems. In BBGC, there is a one-way, one-time migration from Blue to Green. After much preparation, ultimately a single Production Cutover completes the project.

Chapter 2. Migration Planning

2.1. Build a Cross Functional Migration Team

As the Start and End States of the migration are defined, it should become clear what human resources and teams, both internal and external to your organization, are to be assembled. This may include Perforce Helix administrators, system administrators, storage architects, build engineers, some representative users (perhaps from different time zones), and Perforce consultants. Also include folks in your organization needed to implement things like DNS changes or load balancer redirection that may be needed as part of the eventual Production Cutover.

Migration-Style Upgrade projects involve personnel who are necessary or helpful to successful completion of the project, but whose primary job is something other than the migration. It may be helpful to discuss schedule and availability of various resources for the expected duration of the project. In particular, it is important to ensure key resources are available at critical points in the schedule, including standing up Dry Runs and the eventual cutover.

In addition to folks needed to drive the migration, additional resources may be needed for testing, especially for things that need a human eyeball. For example, if your BBGC involves a perfmerge with Helix Swarm, the validation that things "look right" after a Dry Run is best done by someone who uses that software regularly, and has a good sense for what "normal" looks or feels like. Other things, like archive file verification, can be verified mechanically.

Migration projects may benefit from having a project manager and/or an executive sponsor to ensure the migration project priorities are aligned with other organizational efforts.

Designate communication methods for the project. Start a new discussion thread specific to the migration project. This can be an email thread, a dedicated channel in Slack/Teams/Discord, etc. Include relevant stakeholders. Each Dry Run and eventually the Production Cutover should have its own channel.

2.2. Define Start State

The starting environment can pretty much anything:

- Any P4D version going back to 1995.
- P4D server machine(s) operating on Windows, UNIX, Mac OSX, Linux, or other platform.
- Any legacy method of managing Helix Core:
  - An older version of SDP (prior to 2020.1).
  - Home-grown custom scripts, which may be simple or extensive.
  - Management scripts provided by a 3rd party such as ICManage.
  - The p4dctl service, possibly installed with the helix-p4d package installation.
  - Manual procedures.
  - No management whatsoever.

2.3. Take Inventory

At the outset of a Migration Style Upgrade, take stock of everything that comprises the current Helix Core ecosystem. The inventory should be comprehensive of everything that is part of your infrastructure, regardless of whether you intend for it to be affected by the current upgrade project. In the simplest case, inventory might consist of single server machine with a single p4d commit server.

Some items to include in the inventory are:

- All server machines involved in the ecosystem, including those running Helix Core software (such as p4d servers) as well as those not operating any Helix Core services but from which automation runs, such as build server farms.
- All Helix Core software components, such as
  - Server products p4d/p4broker/p4p
  - p4broker p4p, Helix Swarm, P4DTG, Helix DAM, etc.
- Any customization done using Helix Core custom features:
  - Any custom Helix Core triggers. Triggers in Helix Core are a means of customizing and extending Helix Core behaviors.
  - Any custom Helix Core broker scripts.
- Any other custom automation.
- Any systems integrations with 3rd party or home-grown systems, such as issue/bug trackers, task and agile project management tools, and reporting systems. If in doubt about whether a system is potentially affected by or involved in the upgrade, include it in the inventory for consideration.

Any custom elements in the ecosystem, such as triggers, require special consideration when planning migrations. Often you want custom elements to survive the migration, though this may require specialized expertise. That may in turn require adding folks to the migration team who are or can get up to speed on the custom elements. In some cases, it may be worth considering whether there is a need for the custom elements to survive the migration.

If an ecosystem has production and non-production elements (such as a copy-of-production sandbox environment), those distinctions should be noted in the inventory. Such distinctions may effect whether the elements are in the scope of the effort, and when they are handled in the schedule.

2.4. Define End State

At a minimum, the desired **End State** of a Migration-Style Upgrade is as defined above in Section 1.1, “Optimal Helix Core Operating Environment”. In addition to those aspects, you may choose to add other aspects to your desired **End State**, depending on the goals of your migration. In the mindset of "While we’re at it," some common examples things added to the End State definition are:

- **Authentication and Single-Sign On (SSO)**: Enable a Single Sign On (SSO) solution using the Helix Authentication Service.
- **Monitoring**: Deploy monitoring with P4Prometheus.
- **Helix Core Data Consolidation (Perfmerge)**: In some cases, one driving goal of the BBGC is to combine formerly silo'd/independent data sets into a single, larger Helix Core data set, with files and history from both server instances. This is often done to enable more collaborative workflows across teams. This complex process can be achieved by doing a perfmerge, essentially a neurosurgery operation on a set of Helix Core data sets. The complexity and risk of a perfmerge can be mitigated with a Big Blue Green Cutover, which provides the needed infrastructure to do the extensive testing required.
- **Hardware Core Hardware Consolidation**: If the intent is to redeploy Helix Core onto a smaller number of server machines to reduce the hardware footprint of Helix Core, the changes to which data sets operate from which machines can be handled as part of the migration plan. Hardware consolidations are vastly simpler than a data consolidation (ala perfmerge), but still require testing.
- **Hardware Core Topology Expansion**: In some cases a goal of a BBGC is to expand the topology. Merely adding topology components to an existing Helix Core ecosystem, such as adding an edge server or Helix Swarm, does not by itself warrant a BBGC. However, expanding the topology is a common "add on" to a migration plan when the need for doing a BBGC is already established, e.g. due to OS upgrades.

Even if you are near the definition of optimal, say on Rocky Linux 8 but otherwise modern, the Migration-Style Upgrade is the preferred method of getting to the modern topology (on Rocky 9). (If your environment is optimal in all other respects and *only* the P4D version is older, than you might consider an in situ upgrade). Other options are not discussed in this document, because the Migration-Style Upgrade has significant benefits that often make it preferable even when other options are possible.

The specific starting environment will impact migration options and procedures, as will be called out in this document.

Some *big changes* that call for a Migration-Style Upgrade include:

- Windows to Linux migration. Those are discussed in the SDP Windows to Linux Migration Guide.
- Upgrade of a major operating system version, e.g. CentOS 7 → Rocky 8, RHEL 8 → RHEL 9, Ubuntu 20.04 → Ubuntu 22.04, or even a Linux family change, e.g. CentOS 7 → Ubuntu 22.04.
- Upgrade of SDP from a version prior to 2020.1, where the SDP Legacy Upgrades applies, a well-documented but manual upgrade procedure. That document is for in situ upgrades of SDP — this document avoids the in situ procedure entirely by using a Migration-Style Upgrade.

The primary scenario this document is focused on is a migration to a cloud provider, though only minor adaptations are needed if going to an on-premises environment. We lean toward a cloud environments for documentation purposes, because we can assume more, and define more, with a cloud as a target. If your target environment is on-prem, there is a greater likelihood of local polices, practices, and perhaps even different teams within your organization that may be involved.

2.5. Define Project Scope

Scoping the migration project starts with the inventory. Decide on a per-item basis whether that item in the inventory is to be included in the migration, left alone, or perhaps decommissioned. Pay attention to whether any software components are obsolete or have been sunset.

For example, if P4Web (a legacy server that was sunset in 2012) is part of your inventory in the Blue environment, you'll want to plan to replace it with its successor, Helix Swarm. If P4Web was used only by humans, this could be as simple as adding Helix Swarm to the Green infrastructure. If instead P4Web was used as part of some external integration, then the migration plan may include a subproject to perhaps replace P4Web URLs or API calls with Helix Swarm equivalents.

In addition to scoping based on inventory, the scope must take into account considerations mentioned in the End State, such as adding HA/DR, changing the authentication mechanism, doing a perfmerge, etc.

2.6. Plan One or more Dry Runs

The overall project migration plan must account for at least one, and potentially several Dry Runs. A Dry Run exercises parts of the cutover procedure that bring the Green environment to life, but leaves out those parts of the cutover procedure that would bring all users over to the Green environment. Once a Dry Run is completed, the Green environment is online and available for testing.

Dry runs are generally tagged with an identifier, usually just a number, e.g. Dry Run 1, Dry Run 2, etc. Dry runs can be formal or informal. Informal dry runs are those available only to a subset of the Migration Team, typically the Perforce admins or someone responsible for crafting and testing the Cutover Procedure. For formal dry runs, the availability of the Green environment is announced to the Migration Team. This is to simplify testing and coordination of testing resources in the environment. For example, representatives from the user community on the Migration Team can focus their test efforts knowing that the Green environment is expected to be ready to use.

Typically, only one Dry Run can be operational at a time, as Dry Runs use the Green environment.

In the simple case, there will be only one Dry Run, and thus it will be formal. Multiple Dry Runs should be considered if any of the following are true:

- If the Green infrastructure will be delivered in phases, earlier Dry Runs can test those Green components that are available sooner.
- Orchestration automation needs to be tested. One or more Dry Runs (often informal ones) may be dedicated to the purpose of iterating on the orchestration automation.

In addition to planned Dry Runs, sometimes learnings from one Dry Run necessitate additional Dry Runs that may not have been planned at the beginning. Expect the unexpected when it comes to Dry Run planning.

2.7. Plan Orchestration Automation

BBGC migrations often involve automation to orchestrate some steps in the cutover procedure. Done correctly, automation of these "run once" procedures can have significant benefits:

- Shortening the duration of the production cutover procedure by removing human lag between any long-running operations in the cutover procedure, such as operations involving checkpoints.
- Improving the auditability of key aspects of the process by capturing exact commands executed and their outputs, to support diagnostics in event of a failure of some kind. Hopefully issues are discovered and resolved during the dry runs, but having a clear audit trail can be especially valuable in the production cutover.
- Improving visibility and transparency of the actual commands executed, which can have training value and allow for collaboration, team cross-training, and peer oversight. Seeing actual commands to be executed in code is a more precise way to show the outcome of technical decisions made during migration planning than, say verbal explanation or even documentation. Documentation can quickly become out of date, while code is assured to be current if the procedure relies on execution of the code.
- Reducing opportunities for human error. Often custover procedures have a sequences of operations that must be executed precisely and in a precise order, the kind of thing best done with automation.

Orchestration automation is not necessary for all BBGC migrations. The more complex the migration, the greater the benefit of using automation. Generally a cutover strategy will start with high-level steps, and then evolve to greater detail. During the planning process, it can be decided where automation adds the most value. The automation may be aware of the Dry Run number, and depending on the steps to be automated, may need to be aware of the distinction between a Dry Run and the Production Cutover.

Depending on the scope of the orchestration automation, it may require awareness of the distinction between a dry run and the Production Cutover.

When planning the scope of what to automate in the context of a BBGC, be flexible. While there may be multiple Dry Runs, ultimately, the Production Cutover will be done only once. This gives some flexibility in determining the scope of what to automate.

Don’t be too ambitious with the scope of automation, especially considering the schedule goals of the project and the fact that a BBGC is a "one shot" deal. Automation can reduce the cycle time of Dry Run iterations and ultimately the duration of the Production Cutover procedure, along with other automation benefits as noted above. However, Automation is software development and needs proper testing and iteration. If you’re automating a repetitive process, automate everything that can be automated, but if you’re automating a one-time process, be judicious.

2.8. Define Server Templates

It is a good idea to define templates for server machines in some form. You may require multiple templates depending on the classes of machine you will operate in the Green environment. You will certainly have a Helix Core (p4d) server. You may have a separate template for Helix Proxy server, and another for a Helix Swarm server, etc. A review the inventory of server machines can guide the list of server templates needed.

Some common forms of defining a server template are:

- A written language description (perhaps on a wiki page) that defines key characteristics such as server hardware specifics (RAM/CPU), storage system details, operating system, and the like.
- A "golden image," a labeled virtual image in your preferred virtualization infrastructure.
- A script of some kind that converts a base operating installation into one suitable for Helix Core.
- AWS CloudFormation Templates.
- Azure Resource Manager Templates.
- Terraform.
- The Enhanced Studio Pack (ESP) from Perforce. See Section 2.8.1, “Enhanced Studio Pack”.

2.8.1. Enhanced Studio Pack

The Enhanced Studio Pack (ESP) should be considered if the target environment is Amazon or Azure. ESP is not available for on-prem, and not presently available for other cloud environments. Even in cases where ESP is available, it may not be the best choice. Some cases where ESP is not optimal or would need adjustment afterward include:

- You have a corporate policy that dictates machines must be launched from internally produced baseline virtual machines images. ESP is essentially a set of machine images, but of course not be based on any customer’s unique base images. (In some cases, an exception can be granted because ESP is reasonably well secured, using Security Enhanced Rocky Linux provided by Perforce OpenLogic).
- You plan to use advanced storage solutions like NFS.
- Your `perforce` operating system user is defined in LDAP rather than being local. (Note: We recommend using local accounts when you can for optimal reliability, but using an LDAP or NIS account is required when using NFS to ensure numeric user IDs align across a fleet of server machines).

2.9. Consider Client Upgrades

Generally speaking, for a global topology upgrade, it is a good idea to plan to upgrade Helix Core client software as well as server components.

Helix Core has a powerful feature that allows client and server software to be upgraded independently, so that upgrading the server does force an upgrade of all clients at the same time, and clients can upgrade ahead of the server version as well.

However, the greater the disparity between client and server versions, the greater the risk of issues due to version skew between clients and servers. Further, new clients are going to have the latest security features. Lastly, certain product features and security benefits require clients to be upgraded.

Upgrading clients should be considered during a Migration Style Upgrade. Typically Migration Style Upgrades are done infrequently compared to in situ upgrades, perhaps once every 3 to 6 years, and thus a good time to address client/server version skew.

The following should be considered:

- The `p4` command line client binary. Updating this may involve notifying users how to download it themselves, or may involve an admin updating the binary on a system used by others.
- The P4V GUI client, which may be installed by individual users or provided by admins to users.
- Any software built on any of the Helix Core APIs, such as the C++ API or any of the APIs derived from the C++ API such as P4Perl, P4Python, etc. This may include 3rd party tools. Such software will require recompiling with the new API version, and may possible require code changes and associated testing.

If your environment has custom automation built with the C++ or derived APIs, updating such things will require specialized expertise. If the automation is from a third party, you may need to explore options for getting updated versions from the vendor to match the new Helix Core version, or information about whether an upgrade is needed.

If updating clients becomes a complex endeavour, the risk calculus becomes a bit complex. Changing more complex things at once can increase risk, but so can allowing too much version skew. Think of excessive version skew, say 3+ years, as a form of technical debt: Sooner or later you'll need to make a payment.

It you don't have a good sense for what clients are connecting your Helix Core server, some server log analysis can work wonders. There are various approaches, but the gist is that you can scan a series of p4d server logs covering some span of time (a day, a week, or more), and determine what client programs are connecting to your Helix Core servers over that timeframe.

Chapter 3. Migration Preparation

3.1. Build the Green Infrastructure.

With the End State in mind, build out the Green infrastructure, including all server machines needed, deploying any new storage systems, etc.

3.1.1. Green Infrastructure Setup Timing

You might create the entire green infrastructure at once, or build it out in phases. The longer the infrastructure is available, the more testing can be done prior to it going live — this applies to any individual component of the infrastructure, as well as to the Green infrastructure as a whole. Considerations for phased vs. all at once tend to be schedule and cost. If infrastructure components aren't ready when needed for dry runs, that can delay the schedule. If they're available long before they're needed, that might incur additional costs (especially in cloud environments).

If resources can be spun up quickly from templates, it is OK to defer instantiation of those resources until shortly before they're needed. If spinning up services requires lengthy budget, ordering, acquisition, security review, and/or configuration processes, then spinning up resources sooner can avoid lengthy project delays.

3.2. Develop Test Plans

Two separate test plans are needed for a BBGC. The first test plan, the Dry Run Test Plan, is to be executed during dry run(s), before the Green environment becomes the new production environment. These tests are not tightly constrained by time, and thus allow for anything that needs testing on the new infrastructure. These can be potentially comprehensive, intended to test both the migration process and the new infrastructure.

The second test plan, the Cutover Test Plan, is to be executed during the maintenance window for the cutover. Because it occurs during a period of service outage, the Cutover Test Plan is naturally constrained in duration. It is mostly a subset the Dry Run Test Plan, but with some subtle differences.

Some things to include in test plans are:

- **Access Testing**: The server machines in the Green infrastructure need to be accessible from the same end points as the corresponding servers in the Blue infrastructure (unless there are intended changes in access in the End State).
- **Performance Testing**: Often a BBGC involves deployment of new hardware, new server machines, new storage or networking infrastructure, etc. It is a good idea to include performance testing by doing sea trials on the new machines, putting them under some load. Simple things like NFS mount option misconfigurations will not show up with functional testing, and may go unnoticed unless performance testing is done.

3.3. Plan for Rollback

One of the compelling benefits of a BBGC is that a rollback can be relatively straightforward, since the Blue infrastructure is not affected by the cutover procedure in any way that would interfere with the ability to rollback.

That said, planning must account for what the rollback options are, and in particular determine the point after the rollback is no longer an option. Somewhere in the Cutover Procedure, sometime after the Cutover Test Plan is executed, a formal "Go/No Go" decision is made. After that point, a "Fix and roll forward" approach is used.

At any point during the execution of the Production Cutover, things may not go according to plan. In the event of a mishap of any kind, decisions must be made about whether to attempt a rollback and how to execute. Planning can and should go into how to prepare for a rollback, and even some "war gaming" thought exercises. However, there is a limit to planning for a rollback, because a rollback is inherently a reactive operation. A rollback is executed in response to some kind of external cause. Actual rollback procedures require a detailed understanding of any potential desire to rollback. After understanding the business and/or technical reasons for a rollback, a plan should be crafted for the specific situation.

Appendix A: Operational Guidance

Some basic tips for operating during Dry Runs and the Production Cutover:

- Backup Everything Version
- Capture commands executed and their outputs faithfully.

Chapter 4. Sample Cutover Procedure

The following is a SAMPLE cutover procedure. DO NOT be tempted to use this without adaptation to your environment. Cutover procedures need to take into account the specifics of your organization's planned migration and the Start and End states.

The plan must provide high-level steps for the cutover operation. In some cases, these steps are supplemented with great detail, including actual commands to execute, possibily also including diagnostic commands with expected outputs captured during dry runs and later captured for the Production Cutover. Some plans also include expected duration of each step, with projected start and end times, benefitting from experience executing the dry run(s).

4.1. One Week Before Cutover

The following steps are to be executed approximately one week before the Production Cutover. These steps are non-disruptive to the current production (Blue) environment:

STEP 1. Make sure the commit server and all replica and edge servers have healthy checkpoints and a healthy offline_db.

STEP 2. Run the SDP Health Check to verify: https://swarm.workshop.perforce.com/view/guest/perforce_software/sdp/main/doc/ReleaseNotes.html#_sdp_health_checks

If there are any issues, there is time to address them before the maintenance window for the cutover.

4.2. One Day Before Cutover

The following steps are to be executed approximately one day before the Production cutover. These steps are non-disruptive to the current production (Blue) environment:

STEP 1. Make sure the commit server and all replica and edge servers have healthy checkpoints and a healthy offline_db.

STEP 2. Run the SDP Health Check to verify: https://swarm.workshop.perforce.com/view/guest/perforce_software/sdp/main/doc/ReleaseNotes.html#_sdp_health_checks

Because these steps were executed about a week previously, one would hope the status remains the same. If there are any issues, determine if they can be quickly addressed before the maintenance window, or if the maintenance window needs to be rescheduled.

4.3. Maintenance Procedure for Cutover

The following describes the overall maintenance window procedure for the Big Blue Green Cutover. Upon successful completion of this procedure, the Green environment will become the new Production Environment. The old Blue environment will remain online for no less than 30 days before being decomissioned.

All commands are to be operated as the `perforce` OS user.

STEP 1. Create Upgrade Notes File

First, create a directory on the Green commit server for storing, `~perforce/CutoverNotes/Prod-2024.1`, to contain any text files that may be useful to reference later. (For Dry Runs, use something like `~/CutoverNotes/DryRun4-2024.1`). If necessary/useful, create the same folder on other server machines in the Green or Blue infrastructure.

STEP 2. Disable Alerts.

Disable all alerts for the environment to avoid unnecessary alerts during the maintenance window.

Consider whatever mechanisms are used Prometheus/Victoria Metrics, P4Prometheus, Grafana, DataDog, NewRelic, etc.

STEP 3. Disable Crontabs.

Disable crontabs on all server machines in both the Blue and Green infrastructure.

Save current crontab on each host with the following:

```
cd # Go to to the home directory.
crontab -l
crontab -l > crontab.$USER.$(hostname -s)
```

Then, clear the crontab on each machine with:

crontab -r

STEP 4. Lockout Users.

Lockout users with a combination of methods described in 3A, 3B, 3C, and 3D. The general gist is to ensure no users can access the Blue environment during the maintenance window, so that there is zero chance of data entered in the Blue environment not being carried over into the Green environment during the Production Cutover. (Upon successful completion, the Blue infrastructure remains offline and awaits decomission, removal, or repurposing of the infrastructure).

Do Not Rely on communications to users to guarantee that no users are using the Blue systems during this kind of maintenance. Craft operational procedures that provide a guarantee of no data loss even if users don't receive or read/remember details of the maintenance window schedule.

STEP 4A. Lockout Users with Protections

Save the current Protections table on the commit server:

```
cd
p4 protect -o | grep -v ^# | grep -v ^Update: > ~perforce/CutoverNotes/Prod-
```

```
2024.1/protect.before.p4s
cat  ~perforce/CutoverNotes/Prod-2024.1/protect.before.p4s  # <-- Make sure it looks
right.
```

Use a prepared, trimmed Protections "Lockdown" table that allows only the `perforce` super user.

Load it like so, as perforce on the commit server:

p4 protect -i < $P4TMP/protect_locked_down.p4s p4 protect -o

STEP 4B. User Lockout with Brokers

EDITME

STEP 4C. User Lockout with Firewalls

EDITME

STEP 4D. User Lockout with Private Port

THe gist of this approach is to operate services on a port other than the one used for normal operation. For example, if p4d normally runs on port 1666, run it on port 1777 during maintenane.

EDITME

STEP 5. Prepare for Sanity Checks

Capture BEFORE information.

Capture the outpout of some basic commands before the migration that will provide a sanity check when compared with the output of the same commands after the migration. Some commands to run:

- `p4sanity_check.sh`
- On the commit server: `p4 clients -a|wc -l`
- On each edge server: `p4 clients | wc -l`

During dry runs, capture precise details of commands to run. Sometimes the output of commands changes across p4d versions. For example, `p4 clients|wc -l` run on a commit server may yield different results in a commit/edge topology running on p4d 2018.2 then it shows when running on 2023.1. Test the commands you plan to run for both BEFORE and AFTER scenarios in dry runs.

Write a single script to capture the bits you'd like to have captured for BEFORE and AFTER comparison.

STEP 6. Stop All Services

Stop all p4d, p4broker, and p4p services in both the Blue and Green infrastructure.

The Protections change will block most new traffic from coming in, and stopping the services will drop any active connections, as is needed when starting a maintenance window.

STEP 7. Get Blue Checkpoints

Select the set of Blue servers to get checkpoints from. Start with the commit server, and then add any server that requires a separate checkpoint (ignoring any Blue servers that are to be decomissioned as part of the cutover project). The commit server requires a checkpoint, as does any server that has a data set that differs from the the commit. That includes edge servers and any filtered replicas, but excludes standby (as they are unfiltered by definition). Any edge server whose workspaces are expendable, as may be the case with Build Edges (used only by automation), can be excluded from this list.

Start the Blue p4d servers that require checkpoints (and only the p4d servers; no brokers or proxies).

From the commit server, use p4 commands to access and request checkpoints from all others servers that need checkpoints with commands like these:

```
p4 -p ssl:EdgeServer1:1666 trust -y
p4 -p ssl:EdgeServer1:1666 login -a < $SDP_ADMIN_PASSWORD_FILE
p4 -p ssl:EdgeServer1:1666 pull -lj
p4 -p ssl:EdgeServer1:1666 pull -ls
p4 -p ssl:EdgeServer1:1666 pull -ls
```

The get the checkpoints going with:

```
p4 admin journal
```

When those checkpoints are done, stop all services.

STEP 8: Stop All Services

EDITME

STEP 9: Copy checkpoints from all Blue servers to corresponding Green servers.

For edge servers, the Green servers will exist near the corresponding Blue servers, so checkpoint copies will be "local-ish."

EDITME

STEP 10: Load Checkpoitns on Green Servers.

The SDP `load_checkpoint.sh` script will load checkpionts, upgrade data sets, and start services.

EDITME

STEP 11: Execute Sanity Tests in Green Environment.

Exectue the Cutover Test Plan. The cutover plan will start with tests that can be done by administrators, such as archive verification of recent submits, before getting others to spend time doing more detailed testing.

Compare output of comamnds captured earlier.

Note that the Test Plan may require incremental opening of the Protections table to allow testing in the Green environment.

STEP 12: Decide: Go/No Go.

Decide to go forward, or else rollback if needed. The specific rollback steps depend on nature and time of failure.

STEP 12A: Go!

Continue with the plan.

STEP 12B: Abort!

Enable crontabs and alerting in the Blue Environment. Schedule a future date for another attempt.

Plan a retrospective to learn what happened in detail, and how to ensure things go more smoothly next time around.

STEP 13. Start all Green services.

STEP 14. Direct Traffic to the Green Environment.

Redirect user traffic from the the Blue to the Green envirionment. This may involve a mix of DNS changes, load balancer target changes, changes ((hacks?) to `/etc/hosts` files, NIS changes, or even giving out new IP addresses to end users (to be avoided if at all possible).

STEP 15. Restore User Access

Restore the normal Protections table:

```
cd
p4 protect -i < ~perforce/CutoverNotes/Prod-2024.1/protect.before.p4s
```

Depending on what actions were taken to lockout users, make whatever changes in the Green environment are needed to let folks in the front door.

STEP 16. Restore Crontabs

Restore crontabs on Green servers (only).

EDITME

STEP 17. Celebrate

The maintenance is complete. Enjoy a beverge, have a party, or do whatever you do for celebration in your corporate culture.

Appendix B: 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.