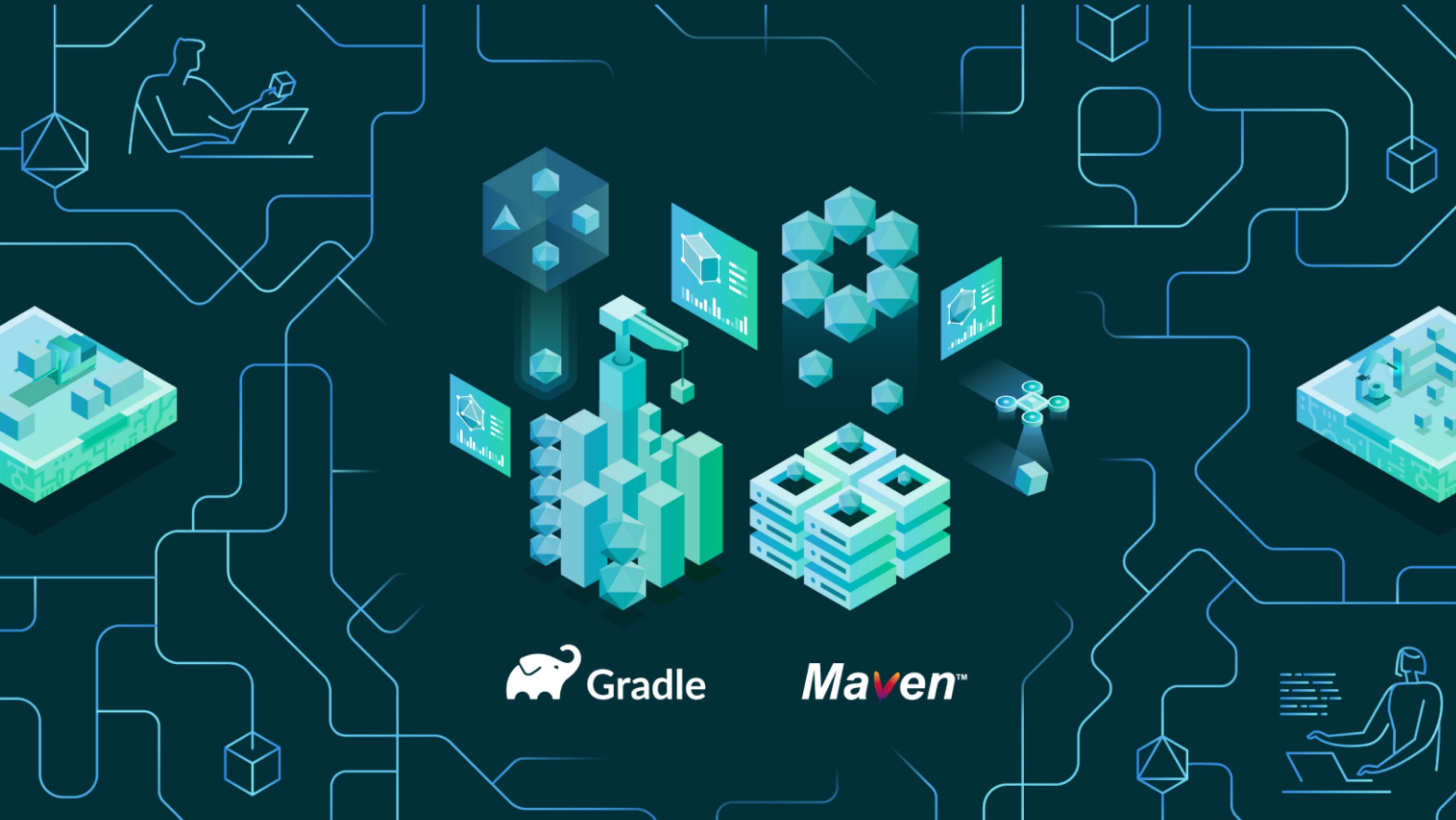
Gradle Enterprise

Customer onboarding

Technical considerations



Welcome to Gradle Enterprise

Welcome to the family of Gradle Enterprise customers. We are excited that you have chosen Gradle Enterprise as a key component in your journey to improve and maintain developer productivity.

We want you to get the most out of running Gradle Enterprise in production for your entire engineering team. Shifting to production usage is a good point in time to revisit and tighten your setup and configuration in terms of resilience, scalability, and security.

Verifying host and application configuration

Hosting hardware and network configuration

Now that Gradle Enterprise will be a critical piece of your development infrastructure, it is a good time to review the hardware and network configuration, to ensure it is ready for production use. The administration manuals for Docker-based and Kubernetes-based installations provide a good basis to review server CPU and memory, storage capacity, and network configuration. If you have questions about how best to configure your production server, our support team is happy to help.

User management

In order to facilitate the rollout of Gradle Enterprise to all your engineers, we recommend you connect Gradle Enterprise to your in-house identity provider. This integration enables more scalable onboarding and offboarding of users, enforcing custom password policies, possibly leveraging 2FA, and convenient configuration of Gradle Enterprise permissions in either your identity provider or directly in Gradle Enterprise. You can learn more about access control and user management in the administration manuals for Docker-based and Kubernetes-based installations.

Most users do not require administrative functions in Gradle Enterprise. If anonymous viewing of build scans is turned off, it will be sufficient to grant them the *Build scan view* permission. If you have turned off anonymous publishing of build scans, the *Build scan publish* permission will also be required. Find more recommendations on build scan access control and build cache access control in the following sections.

Build scan access control

If the Gradle Enterprise server is not available on the public internet, you may choose to allow anonymous viewing and/or publishing of build scans. *Anonymous viewing* facilitates collaboration since all your engineers can open any build scan link without requiring a Gradle Enterprise account and permissions. *Anonymous publishing* facilitates the capture of build scans for all builds since no access key is required for local developer builds and CI builds to publish build scans. Generally speaking, the fewer permissions required for viewing and publishing build scans, the easier it is for your engineering team to adopt and leverage Gradle Enterprise.

If your Gradle Enterprise server is available on the public internet and you are not an OSS project, you must disable all anonymous access to Gradle Enterprise. Disabling *anonymous viewing* means that anyone attempting to open a build scan link requires an account with adequate permissions. Disabling *anonymous publishing* means that in order to publish build scans, all engineers running a build will have to generate and install an access key on their local workstation and all CI builds also need to be provided with an access key.

Build scan data retention & low disk space protection

To ensure you never run out of disk space, we highly recommend configuring build scan data retention. Based on your trial experience, you should have a good idea of how many days of build scans you can retain. You can either adjust the data retention thresholds to match your maximum disk space, or increase your disk space to be able to retain more days of build scans.

We also recommend that you configure all of the low disk space protection features such that you get alerts on low disk space and automatic build scan deletion and ingest rejection when disk space gets critically low.

We also recommend revisiting your Gradle Enterprise disk configuration and ensuring you have the necessary disk throughput to guarantee smooth operations even during peak build times. Our support team is happy to help you figure out your optimal setup.

Backups & Maintenance

We highly recommend that you configure regular backups of Gradle Enterprise. Backups will allow you to recover historic build scan data in case of a disaster recovery incident, likely caused by the database running out of disk space. In case of a Docker-based installation, we recommend that you store the backups on a different volume.

Make sure that the scheduled backups and the scheduled daily maintenance work run at different times of the day. Ideally, these background tasks should run at times of low system load.

Email notifications

To receive notifications on completion of backups and when free disk space is low, you must configure the coordinates of your email server. Early warning is vital in the case of failed backups and low disk space in order to avoid running into any nonrecoverable state.

Make sure you verify that your email server configuration is working by clicking the *Test* button.

Help contact

Configure a help contact that is displayed to your users in the unlikely event that they end up on a help or error page. This way, their concerns and issues can be addressed quickly.

Optimizing build cache setup

Place a primary build cache node close to where CI builds run

CI builds typically both read from and write to the remote build cache, so it is important that the corresponding build cache node is located as close as possible to where your CI builds run. Ideally, CI agents and their build cache node will be in the same network to minimize overhead caused by increased latency and reduced artifact download/upload speed.

If you are running the Gradle Enterprise server close to where the CI builds run, you can use the built-in build cache node that comes pre-installed with Gradle Enterprise. If this is not the case, we advise that you disable access to the built-in build cache node, and instead spin up a separate build cache node in the same network as where your CI builds run, as described in the next section.

Place secondary build cache nodes close to where local builds run

As you roll out the usage of Gradle Enterprise to multiple projects and multiple teams, you may see high remote build cache overhead for builds - likely local developer builds - that run far from the build cache node populated by your CI builds. In such scenarios, it is advisable to set up secondary build cache nodes that, from a network perspective, are closer to where these local developer builds are being executed.

If you operate multiple build cache nodes, we recommend having conditional logic that chooses the best build cache node based on environmental settings. For example, you could check local builds for the locale and set the remote build cache address as a function of the given locale.

Gradle Enterprise does not place any limit on the number of build cache nodes you want to run. You can register additional build cache nodes in the Build Cache Administration view. See the Gradle Enterprise Administration manual for more details.

Enable replication and preemptive replication

When running multiple build cache nodes, it is usually beneficial to enable replication, and particularly preemptive replication. If you have secondary build cache nodes in your infrastructure, these should replicate from the primary build cache node that is populated by CI. Replication avoids unnecessary task re-execution by reusing build artifacts stored in the parent build cache node.

Preemptive replication is a further optimization that removes the overhead of on-demand checking for and downloading of build artifacts from the parent build cache node. With preemptive replication, any new build artifacts from the parent build cache node are immediately pushed to its child build cache nodes.

You can configure the replication settings in the Build Cache Administration view. See the Gradle Enterprise Administration manual for more details.

Protect remote build cache access

In general, only builds running on CI should write to the remote build cache, with the stored build artifacts being available for consumption to all local developer builds and CI builds. To ensure this, write access to the build caches nodes must be password protected, with the credentials only available on CI and injected via environment variables. You may also decide to password protect read access to the build cache nodes, depending on how much you require to lock down access to the build artifacts.

By default, build cache nodes permit anonymous read and write access. This includes the built-in build cache node that comes pre-installed with Gradle Enterprise as well as any standalone build cache nodes that you may install. It is important to verify that access to all your build cache nodes is properly secured, particularly when your build cache is publicly reachable.

Build cache access control is configured on a per-node basis, and is separate from the access control configuration for viewing and publishing build scans. See the Gradle Enterprise Administration manual for more details.

Verifying build configuration

It is important to verify that each connected project is optimally configured in order to get the most value out of Gradle Enterprise. We have collected some recommendations on the settings for build tool, build scan publishing, and build caching that are optimal in most scenarios.

You can find a complete example of a typical production configuration for both Gradle and Maven in our public Github repository.

Review each of the following recommendations in the context of a build scan published by a CI build, as well as a build scan published by a local developer build.

Ensure optimal build tool configuration

Navigate to the Switches view of the build scan. The following build tool settings should typically be enabled for local and CI builds:

Gradle:

- **Build Cache**
- **Gradle Daemon**
- **File system watching**
- **Parallel task execution**

Maven:

- **Max threads > 1 (see Infrastructure view)**

Depending on the hardware of your developers' machines and your CI agents, you might configure different degrees of parallelism for your local developer builds versus CI builds.

Ensure optimal build scan publishing configuration

Navigate to the Switches view of the build scan. The following build scan publishing settings should typically be enabled for local and CI builds:

Gradle and Maven:

- **Task/goal inputs file capturing**
- **Background build scan publication (should be enabled locally, but not on CI)**

Ensure that in your actual build the Gradle Enterprise plugin or extension is configured to *always* publish a build scan for every build.

Ensure optimal build caching configuration

Navigate to the Build Cache section in the Performance view of the build scan. The following build cache settings should typically be enabled:

- Local cache:
  - **Push enabled**
- Remote cache (HTTP):
  - **Push enabled (should be enabled on CI, but not locally)**
  - **Authenticated true (should be true on CI, but not locally)**

With the above configuration, local and CI builds will read from and write to the local build cache. If your CI agents do not retain any state between build invocations, then the local cache will never be available to benefit a later build invocation. In this case, it is appropriate to disable local cache usage on CI.

Local and CI builds will read from the remote build cache, while only CI builds will write to the remote build cache, guarded by authentication and authorization.

Ensure build scans are published for all builds

Navigate to the build scan list and ensure you are capturing build scans for all local and CI builds, from all users, and from all branches. Capturing a build scan for every build run in your engineering team will provide you with a comprehensive picture when reasoning about how your software is built, and allows you to make informed, data-driven decisions.

Once you capture all builds, you can always narrow down the set of builds you want to investigate through the filtering functionality available in all dashboards, including filtering by custom tags and custom values.

Verifying CI configuration

Inject credentials via environment variables

Passing remote build cache credentials and build scan publishing access keys via system properties or command line arguments risks exposing these secrets via the process command line. Instead, we recommend you set these values via environment variables and read them from within your build. This advice is not limited to Gradle Enterprise related credentials, but applies to all secrets passed to the build invocation.

Verify how you pass in credentials to your builds on CI, and ensure all credentials are provided and consumed via environment variables.

Surface build scan links on the CI build summary view

Investigating a CI build failure will often start with the build summary page of your CI server. While the build log captured by CI will contain the build scan link towards the end of the log, navigating to the log view of a failed CI build and then scrolling to the end of the log is cumbersome. It is much more productive for your engineering team to see the build scan link prominently in the CI build summary view, and to navigate straight to the build scan with a single click in order to investigate the build failure in-depth.

Make navigation from CI builds to build scans as frictionless as possible for your engineering team by applying the Jenkins Gradle Plugin or the TeamCity Build Scan Plugin, both available under OSS terms.

Connecting all projects to Gradle Enterprise

While adoption of Gradle Enterprise usually starts with a single project to deeply familiarize yourself with the product, you ultimately want all your projects to be connected to Gradle Enterprise. While there are several ways to go about it, we have collected some advice on how to best achieve this based on our experience from many customers successfully rolling out Gradle Enterprise.

Extract build configuration into a plugin or extension

Within your engineering team, you likely have more than one project that can benefit from Gradle Enterprise. In order to share your Gradle Enterprise build configuration across multiple projects, this configuration is best extracted into its own Gradle plugin or Maven extension and deployed to your internal artifact manager, typically Nexus or Artifactory.

You can find complete examples of a custom Gradle Enterprise Gradle plugin and a custom Gradle Enterprise Maven extension in our public Github repository. It is likely that you have already been using these ready-to-use artifacts as part of your trial. You can copy the examples, fine-tune them to your needs, and publish the resulting plugin or extension to your internal artifact manager.

Roll-out build configuration to all projects

Once you have extracted the build configuration into a custom Gradle plugin or Maven extension and deployed it to your internal artifact manager, your current and future projects can instantly connect to Gradle Enterprise by applying the custom Gradle plugin or Maven extension next to the core Gradle Enterprise Gradle plugin or core Gradle Enterprise Maven extension. Each project can upgrade the core Gradle Enterprise plugin or extension as well as your custom Gradle plugin or Maven extension independently and at the project's own discretion.

You can find an example of how to automate the roll-out of the core and custom Gradle Enterprise Maven extensions in our public Github repository. You can take that script as a possible starting point and adjust it to your needs.

Collaborating on build issues with build scans

During your trial, you spent a significant amount of time analyzing build behavior through build scans in order to optimize performance and reliability. Build scans can also be used on a daily basis for investigating all manner of build issues either by oneself or through collaboration with others.

Whenever they face build issues and require help from others, encourage your team to share build scans with each other – all parts of the build scan can be deep-linked into by copying the URL. Until the team gets into the habit, you may want to establish a rule that all build support requests be accompanied by a build scan.

Observing trends to capture regressions

During your trial, you optimized your local and CI builds to make them faster and more reliable. You can periodically use the various Gradle Enterprise dashboards to monitor whether the optimizations are still holding up or whether some elements have regressed and need to be acted upon accordingly.

Areas that deserve constant close attention are the Failures dashboard and the Tests dashboard. On the Failures dashboard, build masters can find frequently occurring non-verification build failures and look into fixing them. Using the Tests dashboard, developers can see the most failing and flaky tests and easily prioritize their stabilisation.

If you have a weekly recurring team meeting, this is often a good opportunity to look at the different dashboard data and discuss potential follow-up actions. Remember that you can bookmark each dashboard view including its active build scans filter. Use a relative date range filter to ensure you will always be looking at the most recent data.

Let us know if you need any help with interpreting the aggregate data shown on the Gradle Enterprise dashboards. Our Solutions engineers will be happy to help you.

Checkpoint

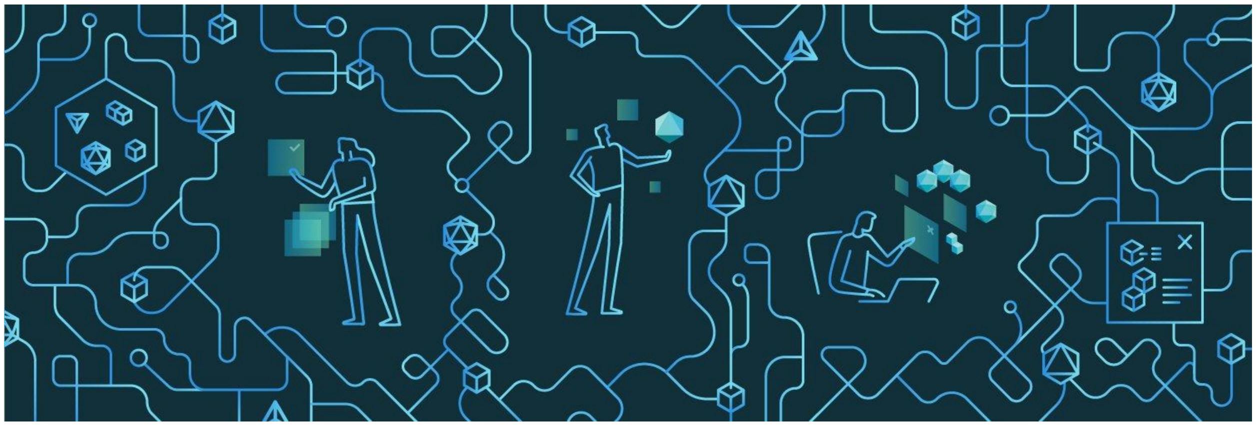
In this document, we have covered technical considerations when running Gradle Enterprise in production. Make sure the following items have been done successfully:

✔ **You have received your Gradle Enterprise production license**

✔ **You have ensured all Gradle Enterprise settings have been adjusted for production**

✔ **You have ensured your server is running in an environment suitable for production usage**

✔ **You have sent us a support bundle via Zendesk so we can ensure the server is healthy**



Version from April 27th 2021