How To...

In this section:

Integrate with an Issue Tracker

TeamCity comes with dedicated support for YouTrack, Jira and Bugzilla.
For any other tracker, you can turn any issue tracker issue ID references in change comments into links. Please see Mapping External Links in Comments for configuration instructions.

Install Multiple Agents on the Same Machine

See the corresponding section under agent installation documentation.

Watch Several TeamCity Servers with Windows Tray Notifier

TeamCity Tray Notifier is used normally to watch builds and receive notifications from a single TeamCity server. However, if you have more than one TeamCity server and want to monitor them with Windows Tray Notifier simultaneously, you need to start a separate instance of Tray Notifier for each of the servers from the command line with the `/allowMultiple` option:

- From the TeamCity Tray Notifier installation folder (by default, it's `C:\Program Files\JetBrains\TeamCity` run the following command:

  ```
  JetBrains.TrayNotifier.exe /allowMultiple
  ```

Optionally, for each of the Tray Notifier instances you can explicitly specify the URL of the server to connect using the `/server` option. Otherwise, for each further tray notifier instance you will need to log out and change server's URL via UI.

```
JetBrains.TrayNotifier.exe /allowMultiple /server:http://myTeamCityServer
```

See also details in the issue tracker.

Move TeamCity Installation to a New Machine

If you need to move existing TeamCity installation to a new hardware or clean OS, it is recommended to follow instructions on copying the server from one machine to another and then switch from the old server to a new one. If you are sure you do not need to preserve old server, you can perform move operations instead of copying in those instructions.

You can use existing license keys when you move the server from one machine to another (as long as there are no two servers running at the same time). As license keys are stored under <TeamCity Data Directory>, you transfer the license keys with all the other TeamCity settings data.

A usual advice is not to combine TeamCity update with any other actions like environment or hardware changes and perform the changes one at a time so that if something goes wrong the cause can be easily tracked.

Switching from one server to another
Please note that TeamCity Data Directory and database should be used by a single TeamCity instance at any given moment. If you configured new TeamCity instance to use the same data, please ensure you shutdown and disable old TeamCity instance before starting a new one.

Generally it is recommended to use a domain name to access the server (in agent configuration and when users access TeamCity web UI). This way you can update the DNS entry to make the address resolve to the IP address of the new server and after all cached DNS results expire, all clients will be automatically using the new server.

However, if you need to use another server domain address, you will need:

- Switch agents to new URL (requires updating `serverUrl` property in buildAgent.properties on each agent).
- Upon new server startup do not forget to update Server URL on Administration | Global Settings page.
- Notify all TeamCity users to use the new address

Move TeamCity Agent

Apart from the binaries, TeamCity agent stores it's configuration and data left from the builds it run. Usually the data from the previous builds makes preparation for the future builds a bit faster, but it can be deleted if necessary.
The configuration is stored under `conf` and `launcher\conf` directories.
The data collected by previous build is stored under `work` and `system` directories.

The most simple way to move agent installation into a new machine or new location is to:

- stop existing agent
- install a new agent
- copy `conf/buildAgent.properties` from the old installation to a new one
- start the new agent.

With these steps the agent will be recognized by TeamCity server as the same and will perform clean checkout for all the builds.

Please also review the section for a list of directories that can be deleted without affecting builds consistency.

Share the Build number for Builds in a Chain Build

A build number can be shared for builds connected by a snapshot dependency or an artifact dependency using a reference to the following dependency property: `%dep.<btID>.system.build.number%`.

For example, you have build configurations A and B that you want to build in sync: use the same sources and take the same build number.
Do the following:

1. Create build configuration C, then snapshot dependencies: A on C and B on C.
2. Set the Build number format in A and B to:

```
%dep.<btID>.system.build.number%
```

Where <btID> is the ID of the build configuration C. The approach works best when builds reuse is turned off via the Do not run new build if there is a suitable one snapshot dependency option.

Read more about dependency properties.

Please watch/comment the issue related to sharing a build number TW-7745.

Change Server Port

See corresponding section in server installation instructions.

Make Temporary Build Files Erased between the Builds

Update your build script to use path stored in `${teamcity.build.tempDir}` (Ant's style name) property as the temp directory. TeamCity agent creates the directory before the build and deletes it right after the build.

Retrieve Administrator Password

On the first start TeamCity displays Administrator Setup page. TeamCity installation should always have a user with System Administrator role.

If there is no user account with System Administrator role in the current authentication scheme, you can use `http://<your_TeamCity_server>/setupAdmin.html` URL to setup administrator account.
If there is an administrator account already, the page is not available and you need to remember the administrator account credentials.

The simplest way to change administrator password is to log in as super user which gives access to profile page of any user.

Prior to TeamCity 8.0 other options were applicable:
If you forgot Administrator password and use internal database, you can reset the password using the instructions.
Otherwise you can use REST API to add System Administrator role to any existing user.
And here is an instruction to patch roles directly in the database provided by a user.

Clear Build Queue if It Has Too Many Builds due to a Configuration Error

Try pausing the build configuration that has the builds queued. On build configuration pausing all its builds are removed form the queue.
Also there is an ability to delete many builds from the build queue in a single dialog.

Estimate Hardware Requirements for TeamCity

The hardware requirements differ for the server and the agents.

The agent hardware requirements are basically determined by the builds that are run. Running TeamCity agent software introduces requirement for additional CPU time (but it can usually be neglected comparing to the build process CPU requirements) and additional memory: about 500Mb. The disk space required corresponds to the disk usage by the builds running on the agent (sources checkouts, downloaded artifacts, the disk space consumed during the build; all that combined for the regularly occurring builds).
Although, you can run build agent on the same machine as the TeamCity server, the recommended approach is to use a separate machine (though, it may be virtual) for each build agent. If you chose to install several agents on the same machine, please consider possible CPU, disk, memory or network bottlenecks that might occur. Performance Monitor build feature can help you in analyzing live data.

The server hardware requirements depend on the server load, which in its turn depends significantly on the type of the builds and server usage. Consider the following general guidelines.

- If you decide to run external database at the same machine with the server, consider hardware requirements with database engine requirements in mind.
- If you face some TeamCity-related Performance issues, they should probably be investigated and addressed individually. e.g. if builds generate too much data, server disk system might need upgrade both by size and

speed characteristics.

Database Note:
When using the server extensively, database performance starts to play greater role.
For reliability and performance reasons you should use external database.
Please see notes on choosing external database.
Database size requirements naturally vary based on the amount of data stored (number of builds, number of tests, etc.) An active server database usage can be estimated at several gigabytes of data.

Overview on the TeamCity hardware resources usage:

- CPU: TeamCity utilizes multiple cores of the CPU, so increasing number of cores makes sense. It is probably not necessary to dedicate more than 8 cores to TeamCity server.
- Memory: See a note on memory usage. Consider also that required memory may depend on the JVM used (32 bit or 64 bit). Generally, you will probably not need to dedicate more than 4G of memory to TeamCity server if you do not plan to run more then 100 concurrent builds (agents) and more then 200 online users.
- HDD/disk usage: This sums up mainly from the temp directory usage (<TeamCity home>/temp and OS temp directory) and .BuildServer/system usage. Performance of the TeamCity server highly depends on the disk system performance. As TeamCity stores large amounts of data under .BuildServer/system (most notably, VCS caches and build results) it is important that the access to the disk is fast. (e.g. please pay attention to this if you plan to store the data directory on a network drive).
- Network: This mainly sums up from the traffic from VCS servers, to clients (web browsers, IDE, etc.) and to/from build agents (send sources, receive build results, logs and artifacts).

The load on the server depends on:

- number of build configurations;
- number of builds in the history;
- number of the builds running daily;
- amount of data consumed and produced by the builds (size of the used sources and artifacts, size of the build log, number and output size of unit tests, number of inspections and duplicates hits, size and number of produced artifacts, etc.);
- cleanup rules configured
- number of agents and their utilization percentage;
- number of users having TeamCity web pages open;
- number of users logged in from IDE plugin;
- number and type of VCS roots as well as checking for changes interval for the VCS roots. VCS checkout mode is relevant too: server checkout mode generates greater server load. Specific types of VCS also affect server load, but they can be roughly estimated based on native VCS client performance;
- number of changes detected by TeamCity per day in all the VCS roots;
- total size of the sources checked out by TeamCity daily.

A general example of hardware configuration capable to handle up to 100 concurrently running builds and running only TeamCity server can be:
Server-suitable modern multicore CPU, 8Gb of memory, fast network connection, fast and reliable HDD, fast external database access

Based on our experience, a modest hardware like
Intel 3.2 GHz dual core CPU, 3.2Gb memory under Windows, 1Gb network adapter, single HDD
can provide acceptable performance for the following setup:

- 60 projects and 300 build configurations (with one forth being active and running regularly);
- more than 300 builds a day;
- about 2Mb log per build;
- 50 build agents;
- 50 web users and 30 IDE users;
- 100 VCS roots (mainly Perforce and Subversion using server checkout), average checking for changes interval is 120 seconds;
- more than 150 changes per day;
- the database (MySQL) is running on the same machine;
- TeamCity server process has `-Xmx1100m -XX:MaxPermSize=120m` JVM settings.

The following configuration can provide acceptable performance for a more loaded TeamCity server:
Intel Xeon E5520 2.2 GHz CPU (4 cores, 8 threads), 12Gb memory under Windows Server 2008 R2 x64, 1Gb network adapter, 3 HDD RAID1 disks (general, one for artifacts, logs and caches storage, and one for the database storage)
Server load characteristics:

- 150 projects and 1500 build configurations (with one third being active and running regularly);
- more than 1500 builds a day;
- about 4Mb log per build;
- 100 build agents;
- 150 web users and 40 IDE users;
- 250 VCS roots (mainly Git, Hg, Perforce and Subversion using agent-side checkout), average checking for changes

interval is 180 seconds;
- more than 1000 changes per day;
- the database (MySQL) is running on the same machine;
- TeamCity server process has `-Xmx3700m -XX:MaxPermSize=300m` x64 JVM settings.

However, to ensure peak load can be handled well, more powerful hardware is recommended.

HDD free space requirements are mainly determined by the number of builds stored on the server and the artifacts size/build log size in each. Server disk storage is also used to store VCS-related caches and you can estimate that at double the checkout size of all the VCS roots configured on the server.

If the builds generate large number of data (artifacts/build log/test data), using fast hard disk for storing .BuildServer/system directory and fast network between agents and server are recommended.

The general recommendation for deploying large-scale TeamCity installation is to start with a reasonable hardware while considering hardware upgrade.
Then increase the load on the server (e.g. add more projects) gradually, monitoring the performance characteristics and deciding on necessary hardware or software improvements. Anyway, best administration practices are recommended like keeping adequate disk defragmentation level, etc.

Starting with an adequately loaded system, if you then increase the number of concurrently running builds (agents) by some factor, be prepared to increase CPU, database and HDD access speeds, amount of memory by the same factor to achieve the same performance.
If you increase the number of builds per day, be prepared to increase the disk size.

If you consider cloud deployment for TeamCity agents (e.g. on Amazon EC2), please also review Setting Up TeamCity for Amazon EC2#Estimating EC2 Costs

A note on agents setup in JetBrains internal TeamCity installation:
We use both separate machines each running a single agent and dedicated "servers" running several virtual machines each of them having a single agent installed. Experimenting with the hardware and software we settled on a configuration when each core7i physical machine runs 3 virtual agents, each using a separate hard disk. This stems form the fact that our (mostly Java) builds depend on HDD performance in the first place. But YMMV.

TeamCity is known to work well with up to 200 build agents (200 concurrently running builds actively logging build run-time data). If you need more agents/parallel builds, it is recommended to setup several separate TeamCity instances and distribute the projects between them. We constantly work on TeamCity performance improvements and are willing to work closely with organizations running large TeamCity installations to study any performance issues and improve TeamCity to handle larger loads.

See also the related blog post.

Estimate the Number of Required Build Agents

There are no precise data and the number of required build agents depends a lot on the server usage pattern, type of builds, team size, commitment of the team to CI process, etc.
The best way is to start with the default 3 agents and see how that plays with the projects configured, then estimate further based on that.

You might want to increase the number of agents when you see:

- builds waiting for an idle agent in the build queue;
- more changes included into each build than you find comfortable (e.g. for build failures analysis);
- necessity for different environments.

We've seen patterns of having an agent per each 20 build configurations (types of builds). Or a build agent per 1-2 developers.

Setup TeamCity in Replication/Clustering Environment

TeamCity does not provide specific support for replication/redundancy/high availability or clustering solutions.
However to address fast disaster recovery scenarios it supports active - failover (hot standby) approach: the data that TeamCity server uses can be replicated and a solution put in place to start a new server using the same data if the currently active server malfunctions.

When setting up TeamCity in a replication environment please note that TeamCity uses both database and file storage to save data. You can browse through TeamCity Data Backup and TeamCity Data Directory pages in to get more information on TeamCity data storing.

Basically, both TeamCity data directory on disk and the database which TeamCity uses should remain in a consistent state and thus should be replicated together.

Only single TeamCity server instance should use database and data directory at any time.

Please also ensure that the distribution of the failover/backup server is of exactly the same version as the main server.

See also information on switching from one server to another.

In case of two servers installations for redundancy purposes, they can use the same set of licenses as only one of they is running at any given moment.

Move TeamCity Projects from One Server to Another

At this time there is no dedicated feature to move projects or build configuration from one TeamCity server to another. Please plan your TeamCity deployment in advance with this in mind. Related supported ability is creating a copy of existing server.

Since TeamCity 8.0 it is possible to move settings of a project or a build configuration to another server with simple file copying. For earlier TeamCity versions see the comment.

For the time being it is not possible to transfer builds information (history, artifacts, etc.)

The two TeamCity servers (source and target) should be of exactly the same version (same build).

All the identifiers throughout all the projects, build configurations and VCS roots of both servers should be unique. If they are not, you can change them via web UI.
If entities with the same id are present on different servers, the entities are assumed to be the same. For example this is useful for having global set of VCS roots on all the servers.

To move settings of the project and all its build configuration from one server to another:

From the TeamCity TeamCity Data Directory, copy the directories of corresponding projects (`.BuildServer\config\projects\<id>`) and all it's parent projects to `.BuildServer\config\projects` of the target server.
This moves project settings, build configuration settings, VCS roots defined in the projects preserving the links between them.
If there are same-named files on the target server as those copied, this can happen in case of
a) id match: same entities already exist on the target server, in which case the clashing files can be excluded from copying, or
b) id clash: different entities happen to have same ids. In this case it should be resolved either by changing entity id on the source or target server to fulfill the uniqueness requirement.

The set of parent projects is to be identified manually based on the web UI or the directory names on disk (which be default will have the same prefix).

Note: It might make sense to keep the settings of the root project synchronized between all the servers (by synchronizing content of `.BuildServer\config\projects_Root` directory). For example, this will ensure same settings for the default cleanup policy on all the servers.

Further steps after projects copying might be:

- delete unused data in the copied parent projects (if any) on the target server
- use "server health" reports to identify duplicate VCS roots appeared in result of copying, if any
- archive the projects on the source server and adjust cleanup rules (to be able to see build's history, if necessary)

What is not copied by the approach above:

- pausing comment and user of the paused build configurations
- archiving user of the archived projects
- global server settings (e.g. Maven settings.xml profiles, tools (e.g. handle.exe), external change viewers, build queue priorities, issue trackers). These are stored under various files under .BuildServer\config directory and should be synchronized either on the file level or by configuring the same settings in the server administration UI.
- project association with agent pools
- templates from other projects which are not parents of the copied one. This configuration is actually deprecated in TeamCity 8.0 and is only supported as legacy. Templates used in several projects should be moved to the common parent project or root project.
- no data configured for the agents (build configurations allowed to run on the agent).
- no user-related or user group-related settings (like roles and notification rules)
- no state-related data like mutes and investigations, etc.

Automatically create or change TeamCity build configuration settings

If you need a level of automation and web administration UI does not suite your needs, there are two possibilities:

- change configuration files directly on disk (see more at TeamCity Data Directory)
- write a TeamCity Java plugin that will perform the tasks using open API.

Attach Cucumber Reporter to Ant Build

If you use Cucumber for Java applications testing you should run cucumber with --expand and special --format options. More over you should specify RUBYLIB environment variable pointing on necessary TeamCity Rake Runner ruby scripts:

```
<target name="features">
   <java classname="org.jruby.Main" fork="true" failonerror="true">
     <classpath>
       <pathelement path="${jruby.home}/lib/jruby.jar"/>
       <pathelement path="${jruby.home}/lib/ruby/gems/1.8/gems/jvyaml-0.0.1/lib/jvyamlb.jar"/>
       ....
     </classpath>
     <jvmarg value="-Xmx512m"/>
     <jvmarg value="-XX:+HeapDumpOnOutOfMemoryError"/>
     <jvmarg value="-ea"/>
     <jvmarg value="-Djruby.home=${jruby.home}"/>
     <arg value="-S"/>
     <arg value="cucumber"/>
     <arg value="--format"/>
     <arg value="Teamcity::Cucumber::Formatter"/>
     <arg value="--expand"/>
     <arg value="."/>
     <env key="RUBYLIB"

value="${agent.home.dir}/plugins/rake-runner/rb/patch/common${path.separator}${agent.home.dir
}/plugins/rake-runner/rb/patch/bdd"/>
     <env key="TEAMCITY_RAKE_RUNNER_MODE" value="buildserver"/>
   </java>
 </target>
```

Please, check RUBYLIB path separator. (';' for Windows, ':' for Linux, or '${path.separator}' in ant for safety)
If you are launching Cucumber tests using Rake build language TC will add all necessary cmdline parameters and env. variables automatically.
P.S: This tip works in TeamCity version >= 5.0.

Get Last Successful Build Number

Use URL like this:

```
http://<your TeamCity server>/app/rest/buildTypes/id:<ID of build
configuration>/builds/status:SUCCESS/number
```

The build number will be returned as a plain-text response.
For `<ID of build configuration>`, see Identifier.
This functionality is provided by REST API

Create a Copy of TeamCity Server with All Data

One of the ways to create a copy of the server is to create a backup, then install a new TeamCity server of the same version that you already run, ensure you have appropriate environment configured (see notes below), ensure that the server uses own TeamCity Data Directory and own database and then restore the backup.
This way the new server won't get build artifacts and some other less important data. If you need them, you will need to copy appropriate directories (e.g. "artifacts") from .BuildServer/system from the original to the copied server.

If you do not want to use bundled backup functionality or need manual control over the process, here is a description of the general steps one would need to perform to manually create copy of the server:

1. create a backup so that you can restore it if anything goes wrong,
2. ensure the server is not running,
3. either perform clean installation or copy TeamCity binaries (TeamCity home directory) into the new place (`temp` and `wor`

k subdirectories can be omitted during copying). ⚠ Use exactly the same TeamCity version. If you plan to upgrade after copying, perform the upgrade only after you have existing version up and running.
4. transfer relevant environment if it was specially modified for existing TeamCity installation. This might include:
   - if you run TeamCity with OS startup (e.g. Windows service), make sure all the same configuration is performed on the new machine
   - use the same TeamCity process launching options, specifically check/copy environment variables starting with `TEAMCITY_`.
   - use appropriate OS user account for running TeamCity server process with appropriately configured settings, global and file system permissions
   - transfer OS security settings if required
   - ensure any files/settings that were configured in TeamCity web UI are accessible; put necessary libraries/files inside TeamCity installation if they were put there earlier)
5. copy TeamCity Data Directory. If you do not need the full copy, refer to the items below for optional items.
   - `.BuildServer/config` to preserve projects and build configurations settings
   - `.BuildServer/lib` and `.BuildServer/plugins` if you have them
   - files from the root of `.BuildServer/system` if you use internal database and you do not want to perform database move.
   - `.BuildServer/system/messages` (optional) if you want build logs (including tests failure details) preserved on the new server
   - `.BuildServer/system/artifacts` (optional) if you want build artifacts preserved on the new server
   - `.BuildServer/system/changes` (optional) if you want personal changes preserved on the new server
   - `.BuildServer/system/pluginData` (optional) if you want to preserve state of various plugins and build triggers
   - `.BuildServer/system/caches` and `.BuildServer/system/caches` (optional) are not necessary to copy to the new server, they will be recreated on startup, but can take some time to be rebuilt (expect some slow down).
6. create copy of the database that your TeamCity installation is using in new schema or new database server. This can be done with database-specific tools or with bundled maintainDB tool by backing up database data and then restoring it.
7. configure new TeamCity installation to use proper TeamCity Data Directory and database (`.BuildServer/config/database.properties` points to a copy of the database)

Note: if you want to do a quick check and do not want to preserve builds history on the new server you can skip step 6 (cloning database) and all items of the step 5 marked as optional.

1. ensure the new server is configured to use another data directory and the database then the original server
   At this point you should be ready to run the copy TeamCity server.
2. run new TeamCity server
3. upon new server startup do not forget to update Server URL on Administration | Global Settings page. You will also probably need to disable Email and Jabber notifiers or change their settings to prevent new server from sending out notifications. If you use cloud integration, you might want to disable that as well.
4. if you need the services on the copied server check that email, jabber and VCS servers are accessible from the new installation.
5. install new agents (or select some form the existing ones) and configure them to connect to the new server (using new server URL)

See also the notes on moving the server from one machine to another.

Licensing issues
You cannot use a single TeamCity license on two running servers at the same time, so to run a copy of TeamCity server you will need another license. Copies of the server created for redundancy/backup purposes can use the same licenses as they only should be running one at a time.
You can get time-limited TeamCity evaluation license once from the official TeamCity download page. If you need an extension of the license or you have already evaluated the same TeamCity version, please contact our sales department.
If you plan to run the second server at the same time as the main one regularly, you need to purchase separate licenses for the second server.

Test-drive Newer TeamCity Version before Upgrade

It's advised to try new TeamCity version before upgrading your production server. Usual procedure is to create a copy of your production TeamCity installation, then upgrade it, try the things out and when everything is checked, drop the test server and upgrade the main one.
When you start the test server do not forget to disable Email and Jabber notifiers and change Server URL to the new one.

Choose OS/Platform for TeamCity Server

Once the server/OS fulfills the requirements, TeamCity can run on any system.
Please also review the requirements for the integrations you plan to use, for example the following functionality requires or works better when TeamCity server is installed under Windows:

- VCS integration with TFS
- VCS integration with VSS

- Windows domain logins (can also work under Linux, but may be less stable), especially NTLM HTTP authentication
- NuGet feed on the server (can also work under Linux, but may be less stable)
- Agent push to Windows machines

If you have no preference, Linux platforms may be more preferable due to more effective file system operations and the level of required general OS maintenance.

Final Operating System choice should probably depend more on the available resources and established practices in your organization.

If you choose to install 64 bit OS, TeamCity can run under 64 bit JDK (both server and agent).
However, unless you need to provide more than 1Gb memory for TeamCity, the recommended approach is to use 32 bit JVM even under 64 bit OS. Our experience suggests that using 64 bit JVM does not increase performance a great deal. At the same time it does increase memory requirements to almost the scale of 2. See a note on memory configuration.

Set up Deployment for My Application in TeamCity

1. Write a build script that will perform the deployment task for the binary files available on the disk. (e.g. use Ant or MSBuild for this. For FTP/SSH tasks check the Deployer plugin) You can also use Meta-Runner to reuse a script with convenient UI.
2. Create a build configuration in TeamCity that will execute the build script and perform the actual deployment. If the deployment is to be visible or startable only by the limited set of users, place the build configuration in a separate TeamCity project and make sure the users have appropriate permissions in the project.
3. In this build configuration configure artifact dependency on a build configuration that produces binaries that need to be deployed.
4. Configure one of the available triggers in the deploying build configuration if you need the deployment to be triggered automatically (e.g. to deploy last successful of last pinned build), or use "Promote" action in the build that produced the binaries to be deployed.
5. Consider using snapshot dependencies in addition to artifact ones and check Build Chains tab to get the overview of the builds.
6. If you need to parametrize the deployment (e.g. specify different target machines in different runs), pass parameters to the build script using custom build run dialog. Consider using Typed Parameters to make the custom run dialog easier to use or handle passwords.
7. If the deploying build is triggered manually consider also adding commands in the build script to pin and tag the build being deployed (via sending a REST API request).
   You can also use a build number from the build that generated the artifact.

Further recommendations:

- Setup a separate build configurations for each target environment
- Use build's Dependencies tab for navigation between build producing the binaries and deploying builds/tasks

Related section on the official site: Continuous Deployment with TeamCity

Use an External Tool that My Build Relies on

If you need to use specific external tool to be installed on a build agent to run your builds, you have the following options:

- Check in the tool into the version control and use relative paths.
- Create a separate build configuration with a single "fake" build which would contain required files as artifacts, then use artifact dependencies to send files to the target build.
- Install and register the tool in TeamCity:
  1. Install the tool on all the agents that will run the build. This can be done manually or via an automated script. For simple file distribution also see Installing Agent Tools
  2. Add `env.` or `system.` property into `buildAgent.properties` file (or add environment variable to the system).
  3. Add agent requirement for the property in the build configuration.
  4. Use the property in the build script.
- Add environment preparation stage into the build script to get the tool form elsewhere.

Integrate with Build and Reporting Tools

If you have a build tool or a tool that generates some report/provides code metrics which is not yet supported by TeamCity or any of the plugins, most probably you can use it in TeamCity even without dedicated integration.

The integration tasks involved are collecting the data in the scope of a build and then reporting the data to TeamCity so that they can be presented in the build results or in other ways.

Data collection
The easiest way for a start is to modify your build scripts to make use of the selected tool and collect all the required data.
If you can run the tool from a command line console, then you can run it in TeamCity with a command line runner. This will

give you detection of the messages printed into standard error output. The build can be marked as failed is the exit code is not zero or there is output to standard error via build failure condition.
If the tool has launchers for any of the supported build scripting engines like Ant, Maven or MSBuild, then you can use corresponding runner in TeamCity to start the tool.

You can also consider creating a Meta Runner to let the tool have dedicated UI in TeamCity.

For an advanced integration a custom TeamCity plugin can be developed in Java to ease tool configuration and running.

Presenting data in TeamCity
The build progress can be reported to TeamCity via service messages and build status text can also be updated.

For testing tools, you can report tests progress to TeamCity from the build via test-related service messages or generate one of the supported XML reports in the build and let it be imported via a service message of configured XML Report Processing build feature.

To present the results for a generic report, the approach might be to generate HTML report in the build script, pack it into archive and publish as a build artifact. Then configure a report tab to display the HTML report as a tab on build's results.

A metrics value can be published as TeamCity statistics via service message and then displayed in a custom chart. You can also configure build failure condition based on the metric.

If the tool reports code-attributing information like Inspections or Duplicates, TeamCity-bundled report can be used to display the results. A custom plugin will be necessary to process the tool-specific report into TeamCity-specific data model. Example of this can be found in XML Test Reporting plugin and FXCop plugin (see a link on Open-source Bundled Plugins).

See also #Import coverage results in TeamCity.

For advanced integration, a custom plugin will be necessary to store and present the data as required. See Developing TeamCity Plugins for more information on plugin development.

TeamCity Security Notes

These notes are provided only for your reference and are not meant to be complete or accurate in their entirety.
TeamCity is developed with security concerns in mind and reasonable efforts are made to make the system not vulnerable to different types of attacks.
However, the general assumption and recommended setup is to deploy TeamCity in a trusted environment with no possibility to be accessed by malicious users.
Here are some notes on different security-related aspects:

- man-in-the middle concerns
  - between TeamCity server and user's web browser: It is advised to use HTTPS for the TeamCity server. During login, TeamCity transmits user login password in an encrypted form with moderate encryption level.
  - between TeamCity agent and TeamCity server: see the section.
  - between TeamCity server and other external servers (version control, issue tracker, etc.): the general rules apply as for a client (TeamCity server in the case) connecting to the external server, see guidelines for the server in question.
- user that has access to TeamCity web UI: the specific information accessible to the user is defined via TeamCity user roles.
- users who can change code that is used in the builds run by TeamCity: the users have the same permissions as the system user under which TeamCity agent is running. Can access and change source code of other projects built on the same agent, modify TeamCity agent code, publish any files as artifacts for the builds built on the agent (which means the files can be then displayed in TeamCity web UI and expose web vulnerabilities or can be used in other builds), etc. It is advised to run TeamCity agents under users with only necessary set of permissions and use agent pools feature to insure that projects requiring different set of access are not built on the same agents. Such users can also view all the projects on the server, see details in the corresponding issue TW-24904.
- users with System Administrator TeamCity role: It is assumed that the users also have access to the computer on which TeamCity server is running under the user account used to run the server process. Thus, some operations like server file system browsing can be accessible by the users.
- TeamCity server computer administrators: have full access to TeamCity stored data and can affect TeamCity executed processes. Passwords that are necessary to authenticate in external systems (like VCS, issue trackers, etc.) are stored scrambled under TeamCity Data Directory and can also be stored in the database. However, the values are only scrambled, which means they can be retrieved by the users who have access to the server file system or database.
- TeamCity agent computer administrators: same as "users who can change code that is used in the builds run by TeamCity".
- Other:
  - TeamCity web application vulnerabilities: TeamCity development team makes reasonable effort to fix any significant vulnerabilities (like cross-site scripting possibilities) once they are uncovered. Please note that any user that can affect build files ("users who can change code that is used in the builds run by TeamCity" or "TeamCity agent computer administrators") can make a malicious file available as build artifact that will then exploit cross-site scripting vulnerability. (TW-27206)
  - TeamCity agent is fully controlled by the TeamCity server: since TeamCity agents support automatic updates download from the server, agents should only connect to a trusted server. An administrator of the server

computer can force execution of arbitrary code on a connected agent.

Restore Just Deleted Project

TeamCity moves settings files of deleted projects under `TeamCity Data Directory/config/_trash` directory.
To restore project you should find the directory on the server and move it into regular projects settings directory: `<TeamCity Data Directory>/config/projects` . Also you should remove suffix \.projectN from the directory name.
You can do this while server is running, it should pick up restored project automatically.

Please note that TeamCity preserves builds history and other data stored in the database for deleted projects/build configurations for 24 hours since the deletion time. All the associated data (builds and test history, changes, etc.) is removed during the next cleanup after 24 hours timeout elapses.

The `config/_trash` directory is not cleaned automatically and can be emptied manually if you are sure you do not need the deleted projects. No server restart is required.

Set Up TeamCity behind a Proxying Server

Consider the example:
TeamCity server is installed at URL: http://teamcity.local:8111
It is visible to the outside world as URL: http://teamcity.public:400/tc

1. An internal TeamCity server should work under the same context as it is visible from outside by an external address. To change TeamCity context root from / to /tc: stop TeamCity server; move `<TeamCity home>\webapps\ROOT` directory to `<TeamCity home>\webapps\tc`; start TeamCity server. Now TeamCity is available via http://teamcity.local:8111/tc/.
2. Set up reverse proxy. The settings below ensure requests to http://teamcity.public:400/tc are redirected to http://teamcity.local:8111/tc and the redirect URLs sent back to the clients are correctly mapped by the proxy server:
    - for Apache

```
LoadModule  proxy_module
/usr/lib/apache2/modules/mod_proxy.so
LoadModule  proxy_http_module
/usr/lib/apache2/modules/mod_proxy_http.so
LoadModule  headers_module
/usr/lib/apache2/modules/mod_headers.so

ProxyRequests       Off
ProxyPass           /tc http://teamcity.local:8111/tc
connectiontimeout=240 timeout=1200
ProxyPassReverse    /tc http://teamcity.local:8111/tc
```

    - for Nginx

```
server {
      listen       400;
      server_name  teamcity.public;

      location /tc {
         proxy_pass        http://teamcity.local:8111/tc;
         proxy_set_header  X-Forwarded-For $remote_addr;
         proxy_set_header  Host $server_name:$server_port;
      }
   }
```

Some other Nginx settings must be changed as well:

```
http {
   proxy_read_timeout     1200;
   proxy_connect_timeout  240;
   client_max_body_size   0;
   ....
}
```

Where `client_max_body_size` controls the maximum size of an HTTP request. It is set to 0 to allow uploading big artifacts to TeamCity.

3. The TeamCity server must know the original remote address of the client. This is especially important for agents, because the server tries to establish connection with an agent to check whether the agent is behind a firewall or not. For this you need to add the following into the Tomcat main <Host> node of the `conf\server.xml` file (see also doc):

```
<Valve
  className="org.apache.catalina.valves.RemoteIpValve"
  remoteIpHeader="x-forwarded-for"
  protocolHeader="x-forwarded-proto"
  internalProxies="192\.168\.0\.1"
  />
```

   Where `internalProxies` must be replaced with the IP address of Nginx or Apache proxy server.

If you need to use different protocols for the public and local address (e.g. make TeamCity visible to the outside world as https://teamcity.public:400) or your proxy server does not support redirect URL rewriting, please use the following approach. Setup a proxying server to redirect all requests to teamcity.public:400 to a dedicated port on a TeamCity server (8111 in the example below) and edit <TeamCity home>\conf\server.xml to change the existing or add a new Connector node:

```
<Connector port="8111" protocol="HTTP/1.1"
          maxThreads="200" connectionTimeout="60000"
          redirectPort="400"  useBodyEncodingForURI="true"
          proxyName="teamcity.public"
          proxyPort="400"
          secure="false"
          scheme="http"
          />
```

For HTTPS case, use `secure="true"` and `scheme="https"` attributes.
This is also described in the comment.

Transfer 3 Default Agents to Another Server

This is not possible.

Each TeamCity server (Professional and Enterprise) allows to use 3 agents without any licenses.
It's a "function" of a server: users do not pay for these agents, there is no license key for them, nor are they bound to the server license key.

So, these 3 agents cannot be transferred to another server as they are "bound" to the server instance.

Each TeamCity server allows to connect agents up to number of agent license keys +3

See more on licensing.

Configure Newly Installed MySQL Server

If MySQL server is going to be used with TeamCity you should review and probably change some of its settings. If MySQL is installed on Windows, the settings are located in `my.ini` file which usually can be found under MySQL installation directory. For Unix-like systems the file is called `my.cnf` and can be placed somewhere under `/etc` directory. Read more about configuration file location in MySQL documentation. Note: you'll need to restart MySQL server after changing settings in `my.ini|my.cnf`.

The following settings should be reviewed and/or changed:

InnoDB database engine

Make sure you're using InnoDB database engine for tables in TeamCity database. You can check what engine is used with help of this command:

```
show table status like '<table name>';
```

or for all tables at once:

```
show table status like '%';
```

max_connections

You should ensure `max_connections` parameter has bigger value than the one specified in TeamCity `<TeamCity data directory>/config/database.properties` file.

innodb_buffer_pool_size

Too small value in `innodb_buffer_pool_size` can affect performance significantly:

```
# InnoDB, unlike MyISAM, uses a buffer pool to cache both indexes and
# row data. The bigger you set this the less disk I/O is needed to
# access data in tables. On a dedicated database server you may set this
# parameter up to 80% of the machine physical memory size. Do not set it
# too large, though, because competition of the physical memory may
# cause paging in the operating system.  Note that on 32bit systems you
# might be limited to 2-3.5G of user level memory per process, so do not
# set it too high.
innodb_buffer_pool_size=2000M
```

We recommend to start with 2Gb and increase it if you experience slowness and have enough memory.

innodb_file_per_table

For better performance you can enable so called per-table tablespaces.
Note that once you add `innodb_file_per_table` option new tables will be created and placed in separate files, but tables created before enabling this option will still be in the shared tablespace.
You'll need to re-import database for them to be placed in separate files.

log files on different disk

Placing the MySQL log files on different disk sometimes helps improving performance. You can read about it in MySQL documentation.

Setting The Binary Log Format

If the default MySQL binary logging format is not MIXED (it depends on the version of MySQL you are using), then it should be explicitly set to MIXED:

```
binlog-format=mixed
```

Import coverage results in TeamCity

TeamCity comes bundled with IntelliJ IDEA/Emma and, since TeamCity 8.1, JaCoCo coverage engines for Java and dotCover/NCover/PartCover for .NET.
However, there are plenty of other coverage tools out there, like Cobertura and others which are not directly supported by TeamCity.

In order to achieve similar experience with these tools you can:

- publish coverage HTML report as TeamCity artifact: most of the tools produce coverage report in HTML format, you can publish it as artifact and configure report tab to show it in TeamCity. If published artifact name is `coverage.zip` and there is `index.html` file in it, report tab will be shown automatically. As to running an external tool, check #Integrate with Build and Reporting Tools.
- extract coverage statistics from coverage report and publish statistics values to TeamCity with help of service message: if you do so, you'll see coverage chart on build configuration Statistics tab and also you'll be able to fail a build with the help of a build failure condition on a metric change (for example, you can fail build if the coverage drops).

Percentage values
You should not publish values CodeCoverageB, CodeCoverageL, CodeCoverageM, CodeCoverageC standing for block/line/method/class coverage percentage. TeamCity will calculate these values using their absolute parts. E.g. CodeCoverageL will be calculated as CodeCoverageAbsLCovered divided by CodeCoverageAbsLTotal.
You could publish these values but in this case they will lack decimal parts and will not be useful.

Debug a Build on a Specific Agent

In case a build fails on some agent, it is possible to debug it on this very agent to investigate agent-specific issues. Do the following:

1. Go to the Agents page in the TeamCity Web UI and select the agent.
2. Disable the agent to temporarily remove it from the build grid. Add a comment (optional). To enable the agent automatically after a certain time period, check the corresponding box and specify the time.
3. Select the build to debug.
4. Open the Custom Run dialog and specify the following options:
   a. In the Agent drop-down, select the disabled agent.
   b. It is recommended to select the run as Personal Build option to avoid intersection with regular builds.
5. When debugging is complete, enable the agent manually if automatic re-enabling has not been configured.

You can also perform remote debugging of tests on an agent via the IntelliJ IDEA plugin for TeamCity.

Debug a Part of the Build (a build step)

If a build containing several steps fails at a certain step, it is possible to debug the step that breaks. Do the following:

1. Go to the build configuration and disable the build steps up to the one you want to debug.
2. Select the build to debug.
3. Open the Custom Run dialog and select the put the build to the queue top to give you build the priority.
4. When debugging is complete, re-enable the build steps.

Protect TeamCity against Heartbleed, ShellShock vulnerabilities

TeamCity distributions provided by JetBrains do not contain software/libraries and do not use technologies affected by Heartbleed and Shell Shock vulnerabilities.
What might still need assessment is the specific TeamCity installation implementation which might use the components behind those provided/recommended by JetBrains and which can be vulnerable to the mentioned exploits.

Protect TeamCity against POODLE vulnerability

If you configured HTTPS access to the TeamCity server, inspect the solution used for HTTPS as that might be affected (e.g. Tomcat seems to be affected). At this time none of TeamCity distributions include HTTPS access by default and investigating/eliminating HTTPS-related vulnerability is out of scope of TeamCity.

Depending on the settings used, TeamCity server (and agent) can establish HTTPS connections to other servers (e.g. Subversion). Depending on the server settings, those connections might fall back to using SSL 3.0 protocol. The recommended solution is not TeamCity specific and it to disable SSL v3 on the target, SSL-server side.

There is a known issue when connecting to Subversion servers via https:// protocol when the server (and also agent in case of agent-site checkout) is running under Java 1.6. In this case TeamCity forces SSLv3 protocol. Please run the TeamCity sever and agents under Java 1.7 or (for TeamCity 8.1.x) check a patched plugin with a fix attached to the related issue.