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 the old server data you can probably perform move operations instead of copying.

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 staring 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 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 `Server configuration` administration 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

Suppose you have build configurations A and B that you want to build in sync: use same sources and take the same build number.
Solution:

1. Create a 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 internal ID of the build configuration C. Please refer to the Build Configuration page for description of how to determine build configuration ID.

This reference is also available if you use artifact dependencies instead of snapshot.
Read more about dependency properties.

We plan to provide more option on build number sharing. Please watch/comment on TW-7745.

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.
    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.

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 in the current authentication scheme.
See also notes when switching from one authentication scheme to another.

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 in the current authentication scheme, the page is not available and you need to remember the administrator account credentials.

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.

Related feature requests in our tracker: TW-1964, TW-4524, TW-1681.

How do I clear build queue if it got 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. 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.

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.

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). You will probably not need to dedicate more than 4G of memory to TeamCity server.
- HDD/disk usage: This sums up 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 generated by the builds (size of the build log, number and output size of unit tests, number of inspections and duplicates hits 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.

Based on our experience, a modest hardware like 3.2 dual core CPU, 3.2Gb memory under Windows, 1Gb network adapter 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, main TeamCity process has `-Xmx1100m -XX:MaxPermSize=120m` 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.

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 and add more projects to the server 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.

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.

Setup TeamCity in Replication/Clustering Environment

TeamCity does not provide specific support for any of replication/high availability or clustering solutions; however you can replicate the data that TeamCity server uses and prepare to start a new server using the same data if existing 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 database that 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 backup server is of exactly the same version as the main server.

See also information on switching from one server to another.

Move TeamCity projects from one server to another

Generally, moving projects to a server that already have projects/build configurations configured is not supported. For addressing simple cases manually, please see a comment.

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

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:<internal ID of build configuration>/builds/status:SUCCESS/number
```

The build number will be returned as a plain-text response.
For `<internal ID of build configuration>`, see Build Configuration#Build Configuration ID.
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, 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 `work` 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
    - 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
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 `Server configuration` administration page. You will also probably need to modify Email and Jabber notification sending settings to prevent new server from sending out notifications
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.
You can get TeamCity evaluation license 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.

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.

How do I 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 (e.g. integration with Microsoft TFS and VSS will work only under MS Windows)

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.

How do I 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 for this)
2. Create a build configuration in TeamCity that will execute the build script.
3. In this build configuration configure artifact dependency on a build that produces binaries that need to be deployed
4. Configure one of the available triggers 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 produces the binaries that need to be deployed.
5. 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.
   You can also use a build number from the build that generated the artifact.

Integrating with Reporting/Metric Tools

If you have a tool that generates some report or provides code metrics, you may want to display the data in TeamCity.

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.
For an advanced integration a custom TeamCity plugin can be developed to ease tool configuration and running. See XML Test Reporting and FXCop plugin (see a link on Open-source Bundled Plugins) as an example.

Presenting data in TeamCity
For a report, the most simple approach is 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.

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).

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.