Setting up and Running Additional Build Agents

This page covers:

Before you can start customizing projects and creating build configurations, you need to configure build agents. Please review the agent-server communication and Prerequisites section before proceeding with agent installation.

- If you install TeamCity bundled with a Tomcat servlet container, or use the TeamCity installer for Windows, both the server and one build agent are installed on the same machine. This is not a recommended setup for p roduction purposes because of security concerns and since the build procedure can slow down the responsiveness of the web UI and overall TeamCity server functioning. If you need more build agents, perform the procedure described below.
- If you need the agent to run a operating system different from the TeamCity server, perform the procedure described below.
- For production installations, it is recommended to adjust the Agent's JVM parameters to include the `-server` o ption.

Prerequisites

Necessary OS and environment permissions

Before the installation, please review the Conflicting Software section. In case of any issues, make sure no conflicting software is used.

Please note that in order to run a TeamCity build agent, the user account used to run the Agent requires the following privileges:

Network

- An agent must be able to open connections to the server using the server address specified in the `serverUrl` property (usually the same URL as the server web UI). Specific URLs which can be used for requests to the server should not be limited. See also the recommended reverse proxy settings.

Legacy bidirectional communication notes

By default unidirectional agent-to-server connection via polling protocol is used by TeamCity.

If you need to use legacy bidirectional communication (not recommended), in addition for the agent to server connections, the server must be able to open HTTP connections to the agent. The agent port is determined using the `ownPort` property of the `buildAgent.properties` file as the starting port (9090 by default, next port is used if the specified port is busy), and the following IP addresses are tried:

- the address specified in the `ownAddress` property of the `buildAgent.properties` file (if any)
- the source IP of the HTTP request received by the server when the agent establishes a connection to the server. If a proxying server is used, it must be correctly configured.
- the addresses of the network interfaces on the agent machine

If the agent is behind NAT and cannot be accessed by any of addresses of the agent machine network interfaces, please specify the `ownAddress` property in the `buildAgent.properties` file.

Please ensure that any firewalls installed on the agent, server machine, or in the network and network configuration comply with these requirements.

Common

The agent process (java) should:

- be able to open outbound HTTP connections to the server address (the same address you use in the browser to view the TeamCity UI, configured in `buildAgent.properties` file)
- have full permissions (read/write/delete) to the following directories recursively: `<agent home>` (necessary for automatic agent upgrade and agent tools support), `<agent work>`, `<agent temp>`, and agent system directory (set by workDir, tempDir and systemDir parameters in buildAgent.properties file)
- be able to launch processes (to run builds).
- be able to launch nested processes with the following parent process exit (this is used during agent upgrade)

Windows

- Log on as a service (to run as Windows service)
- Start/Stop service (to run as Windows service, necessary for the agent upgrade to work, see also Microsoft KB article)
- Debug programs (required for take process dump functionality)
- Reboot the machine (required for agent reboot functionality)
- To be able to monitor performance of a build agent run as a Windows service, the user starting the agent must be a member of the Performance Monitor Users group

For granting necessary permissions for unprivileged users, see Microsoft documentation.

You can assign rights to manage services with Microsoft SubInACL utility, a command-line tool enabling administrators to directly edit security information. The tool uses the following syntax:

```
SUBINACL /SERVICE \\MachineName\ServiceName
/GRANT=[DomainName]UserName[=Access]
```

For example, to grant Start/Stop rights, you might need to execute the command:

```
subinacl.exe /service TCBuildAgent /grant=<user login name>=PTO
```

Linux

- the user must be able to run the `shutdown` command (for the agent machine reboot functionality and the machine shutdown functionality when running in a cloud environment)
- when using systemd, it should not kill the processes on the main process exit (use RemainAfterExit=yes)

Build-related Permissions

The build process is launched by a TeamCity agent and thus shares the environment and is executed under the OS user used by the TeamCity agent. Please ensure that the TeamCity agent is configured accordingly.
See Known Issues for related Windows Service Limitations.

Agent-Server Data Transfers

A TeamCity agent connects to the TeamCity server via the URL configured as the "serverUrl" agent property. This is called unidirectional agent-to-server connection. If specifically configured, TeamCity agent can use legacy bidirectional communication which also requires establishing a connection from the server to the agents.
To view whether the agent-server communication is unidirectional or bidirectional for a particular agent, navigate to Agents | <Agent Name> | Agent Summary tab, the Details section, Communication Protocol.

Unidirectional Agent-to-Server Communication

Agents use unidirectional agent-to-server connection via the polling protocol: the agent establishes an HTTP(S) connection to the TeamCity Server, and polls the server periodically for server commands.

It is recommended to use HTTPS for agent to server communications (check related server configuration notes). If the agents and the server are deployed into a secure environment, agents can be configured to use plain HTTP URL for connections to the server as this reduces transfer overhead. Note that the data travelling through the connection established from an agent to the server includes build settings, repository access credentials and keys, repository sources, build artifacts, build progress messages and build log. In case of using http:// protocol that data can be compromised via "man in the middle attack".

Bidirectional Communication

The bidirectional communication is a legacy connection between the agent and the server and it needs to be specifically enabled (see below). When enabled, it requires the agent to connect to the server via HTTP (or HTTPS) and the server to connect to the agent via HTTP.

The data that is transferred via the connections established by the server to agents is passed via an unsecured HTTP channel and thus is potentially exposed to any third party that may listen to the traffic between the server and the agents. Moreover, since the agent and server can send "commands" to each other, an attacker that can send HTTP requests and capture responses may in theory trick the agent into executing an arbitrary command and perform other actions with a security impact.

Changing Communication Protocol

The communication protocol used by TeamCity agents is determined by the value of the `teamcity.agent.communicationProtocols` server internal property. The property accepts a comma-separated ordered list of protocols (`polling` " and "xml-rpc" are supported protocol names) and is set to `teamcity.agent.communicationProtocols=polling` by default. If several protocols are specified, the agent tries to connect using the first protocol from this list and if it fails, it will try to connect via the second protocol in the list. "polling" means unidirectional protocol, "xml-rpc" - older, bidirectional communication.

- To change the communication protocol for all agents, set the TeamCity server `teamcity.agent.communicationProtocols` server internal property. The new setting will be used by all agents which will connect to the server after the change. To change the protocol for the existing connections, restart the TeamCity server.
- By default, the agent's property is not configured; when the agent first connects to the server, it receives it from the TeamCity server. To change the protocol for an individual agent after the initial agent configuration, change the value of the `teamcity.agent.communicationProtocols` property in the agent's properties. The agent's property overrides the server property. After the change the agent will restart automatically upon finishing a running build, if any.
- When connecting TeamCity agents of version 2018.1+ to TeamCity server of version before 9.1 (e.g. after roll back following an upgrade), the agents will need to be updated to use "xml-rpc" protocol (or they can be reinstalled) to be able to connect to an older server to perform the self-update procedure.

Installing Additional Build Agents

1. Install a build agent using any of the following options:
   - Using Windows installer (Windows only)
   - By downloading a zip file and installing manually (any platform)
   - Via the Docker Agent Image option to prepare a Docker container based on the official TeamCity Agent image
2. After installation, configure the agent specifying its name and the address of the TeamCity server in the `conf/buildAgent.properties` file.
3. Start the agent. If the agent does not seem to run correctly, please check the agent logs.

When the newly installed agent connects to the server for the first time, it appears on the `Agents` page, `Unauthorized agents`

tab visible to administrators/users with the permissions to authorize it. Agents will not run builds until they are authorized in the TeamCity web UI. The agent running on the same computer as the server is authorized by default.

The number of authorized agents is limited by the number of agents licenses on the server. See more under Licensing Policy.

Installing via Windows installer

1. In the TeamCity Web UI, navigate to the Agents tab.
2. Click the Install Build Agents link and select MS Windows Installer to download the installer.
3. On the agent, run the `agentInstaller.exe` Windows Installer and follow the installation instructions.

> ⚠ Please ensure that the user account used to run the agent service has appropriate permissions

Installing via Docker Agent Image

1. In the TeamCity Web UI, navigate to the Agents tab.
2. Click the Install Build Agents link and select Docker Agent Image.
3. Follow the instructions on the page that opens.

Installing via ZIP File

1. Make sure a JDK (JRE) 1.8.0_161 or later (Java 6-10 are supported, but 1.8.0_161+ is recommended) is properly installed on the agent computer.
2. On the agent computer, make sure the `JRE_HOME` or `JAVA_HOME` environment variables are set (pointing to the installed JRE or JDK directory respectively).
3. In the TeamCity Web UI, navigate to the Agents tab.
4. Click the Install Build Agents link and select Zip file distribution to download the archive.
5. Unzip the downloaded file into the desired directory.
6. Navigate to the `<installation path>\conf` directory, locate the file called `buildAgent.dist.properties` and rename it to `buildAgent.properties`.
7. Edit the `buildAgent.properties` file to specify the TeamCity server URL (usage of https:// is recommended, see the notes) and the name of the agent. Please refer to Build Agent Configuration section for details on agent configuration.
8. Under Linux, you may need to give execution permissions to the `bin/agent.sh` shell script.

> ⓘ On Windows you may also want to install the build agent windows service instead of the manual agent startup.

Installing via Agent Push

TeamCity provides functionality that allows installing a build agent to a remote host. Currently supported combinations of the server host platform and targets for build agents are:

- from the Unix-based TeamCity server, build agents can be installed to Unix hosts only (via SSH).
- from the Windows-based TeamCity server, build agents can be installed to Unix (via SSH) or Windows (via psexec) hosts.

> ⚠ SSH note
> Make sure the "Password" or "Public key" authentication is enabled on the target host according to preferred authentication method.

Remote Host Requirements

There are several requirements for the remote host:

Platform	Prerequisites

Linux	1. Installed JDK(JRE) 6-10 (1.8.0_161 or later is recommended). The JVM should be reachable with the JAVA_HOME(JRE_HOME) global environment variable or be in the global path (i.e. not in user's .bashrc file, etc.) 2. The `unzip` utility. 3. Either `wget` or `curl`.
Windows	1. Installed JDK/JRE 6-10 (1.8.0_161 or later is recommended). 2. `Sysinternals psexec.exe` on the TeamCity server. It has to be accessible in paths. You can install it using Administration | Tools page. Note that PsExec applies additional requirements to a remote Windows host (for example, administrative share on remote host must be accessible). Read more about PsExec.

Installation

1. In the TeamCity Server web UI navigate to the Agents | Agent Push tab, and click Install Agent....
   If you are going to use same settings for several target hosts, you can create a preset with these settings, and use it each time when installing an agent to another remote host.
2. In the Install agent dialog, either select a saved preset or choose "Use custom settings", specify the target host platform and configure corresponding settings.
   Agent push to a Linux system via SSH supports custom ports (the default is 22) specified as the SSH port parameter. The port specified in a preset can be overridden in the host name, e.g. `hostname.domain:2222`, during the actual agent installation.
3. You may need to download `Sysinternals psexec.exe`, in which case you will see the corresponding warning and a link to the Administration | Tools page where you can download it.

You can use Agent Push presets in Agent Cloud profile settings to automatically install a build agent to a started cloud instance.

Starting the Build Agent

TeamCity build agents can be started manually or configured to start automatically.

Manual Start

To start the agent manually, run the following script:

- **for Windows:** `<installation path>\bin\agent.bat start`
- **for Linux and macOS:** `<installation path>\bin\agent.sh start`

Automatic Start

To configure the agent to be started automatically, see the corresponding sections:
Windows
Linux : configure daemon process with `agent.sh start` command to start it and `agent.sh stop` command to stop it.
macOS

Automatic Agent Start under Windows

To run agent automatically on the machine boot under Windows, you can either set up the agent to be run as a Windows service or use another way of the automatic process start.
Using the Windows service approach is the easiest way, but Windows applies some constraints to the processes run this way.
A TeamCity agent works reliably under Windows service provided all the requirements are met, but is often not the case for the build processes configured to be run on the agent.

That is why it is recommended to run a TeamCity agent as a Windows service only if all the build scripts support this.
Otherwise, it is advised to use alternative OS-specific ways to start a TeamCity agent automatically.
One of the ways is to configure an automatic user logon on Windows start and then configure the TeamCity agent start (via `agent.bat start`) on the user logon (e.g. via Windows Task Scheduler).

Build Agent as a Windows Service

In Windows, you may want to run TeamCity agent as a Windows service to allow it running without any user logged on.
If you use the Windows agent installer, you have an option to install the service in the installation wizard.

Service system account
To run builds, the build agent must be started under a user with sufficient permissions for performing a build and managing the service. By default, a Windows service is started under the SYSTEM account which is not recommended for production use due to extended permissions the account uses. To change it, use the standard Windows Services applet (Control Panel|Administrative Tools|Services) and change the user for the `TeamCity Build Agent` service.

If you start an Amazon EC2 cloud agent as a Windows service, check related notes.

The following instructions can be used to install the Windows service manually (e.g. after .zip agent installation). This procedure should also be performed to create Windows services for the second and following agents on the same machine.

To install the service:

1. Check if the service with the required name and id (see #4 below, service name is "TeamCity Build Agent" by default) is not present; if installed, remove it.
2. Check that the `wrapper.java.command` property in the `<agent home>\launcher\conf\wrapper.conf` file contains a valid path to the Java executable in the JDK installation directory. You can use `wrapper.java.command=../jre/bin/java` for the agent installed with the Windows distribution. Make sure to specify the path of the java.exe file without any quotes.
3. If you want to run the agent under user account (recommended) and not "System", add "wrapper.ntservice.account" and "wrapper.ntservice.password" properties to the `<agent home>\launcher\conf\wrapper.conf` file with appropriate credentials
4. (for second and following installations) modify the `<agent>\launcher\conf\wrapper.conf` file so that the `wrapper.console.title`, `wrapper.ntservice.name`, `wrapper.ntservice.displayname` and `wrapper.ntservice.description` properties have unique values within the computer.
5. Run the `<agent home>\bin\service.install.bat` script under a user with sufficient privileges to register the new agent service. Make sure to start the agent for the first time only after it is configured as described.

To start the service:

- Run `<agent home>/bin/service.start.bat`
  (or use Windows standard Services applet)

To stop the service:

- Run `<agent home>/bin/service.stop.bat`
  (or use Windows standard Services applet)

You can also use the standard Windows `net.exe` utility to manage the service once it is installed.
For example (assuming the default service name):

```
net start TCBuildAgent
```

The `<agent home>\launcher\conf\wrapper.conf` file can also be used to alter the agent JVM parameters.

The user account used to run the build agent service must have enough rights to start/stop the agent service, as described above.

Automatic Agent Start under Linux

To run agent automatically on the machine boot under Linux, configure daemon process with the `agent.sh start` command to start it and `agent.sh stop` command to stop it.

For systemd, see the example configuration file:

teamcityagent.service

```
[Unit]
Description=TeamCity Build Agent
After=network.target

[Service]
Type=oneshot

User=teamcityagent
Group=teamcityagent
ExecStart=/home/teamcityagent/agent/bin/agent.sh start
ExecStop=-/home/teamcityagent/agent/bin/agent.sh stop
```

```
# Support agent upgrade as the main process starts a child and exits then
RemainAfterExit=yes
# Support agent upgrade as the main process gets SIGTERM during upgrade and that maps to exit code 143
SuccessExitStatus=0 143

[Install]
WantedBy=default.target
```

For init.d, refer to an example procedure:

1. Navigate to the services start/stop services scripts directory:

```
cd /etc/init.d/
```

2. Open the build agent service script:

```
sudo vim buildAgent
```

3. Paste the following into the file :

```
#!/bin/sh
### BEGIN INIT INFO
# Provides:          TeamCity Build Agent
# Required-Start:    $remote_fs $syslog
# Required-Stop:     $remote_fs $syslog
# Default-Start:     2 3 4 5
# Default-Stop:      0 1 6
# Short-Description: Start build agent daemon at boot time
# Description:       Enable service provided by daemon.
### END INIT INFO
#Provide the correct user name:
USER="agentuser"

case "$1" in
start)
 su - $USER -c "cd BuildAgent/bin ; ./agent.sh start"
;;
stop)
 su - $USER -c "cd BuildAgent/bin ; ./agent.sh stop"
;;
*)
  echo "usage start/stop"
  exit 1
 ;;

esac

exit 0
```

4. Set the permissions to execute the file:

```
sudo chmod 755 buildAgent
```

5. Make links to start the agent service on the machine boot and on restarts using the appropriate tool:

- For Debian/Ubuntu:

```
sudo update-rc.d buildAgent defaults
```

- For Red Hat/CentOS:

```
sudo chkconfig buildAgent on
```

Automatic Agent Start under macOS

For macOS/Mac OS X, TeamCity provides the ability to load a build agent automatically when a build user logs in.

Login Item approach (recommended)

The recommended way to start build agent on MacOS is as follows:

- Install a build agent on a Mac via `buildAgent.zip`
- Prepare the `conf/buildAgent.properties` file (set agent name there, at least)
- Make sure that all files under the `buildAgent` directory are owned by `your_build_user` to ensure a proper agent upgrade process.
- Start the build agent via `bin/agent.sh start` command and make sure it upgrades successfully on the first start. This may require a couple of minutes, see `buildAgent/logs/teamcity-agent.log` for progress.
- Create a command file `$HOME/start_build_agent.command` which contains one line:

```
/path/to/buildAgent/bin/agent.sh start
```

- Add this command file into Login Items via UI or use the following command from the Terminal app:

```
osascript -e 'tell application "System Events" to make login item at end with properties {path:"/Users/your_build_user/start_teamcity_agent.command", hidden:false}'
```

- Configure your Mac system to automatically login as a build user, as described here
- Reboot
- On the system startup, the build user should automatically log in, and the build agent should start from the Terminal app (you can close it after that).

LaunchAgent approach

To configure an automatic build agent startup via `LaunchAgent`, follow these steps:

- Install a build agent on a Mac via `buildAgent.zip`
- Prepare the `conf/buildAgent.properties` file (set agent name there, at least)
- Make sure that all files under the `buildAgent` directory are owned by `your_build_user` to ensure a proper agent upgrade process.
- Load the build agent via command:

```
Run these commands under your_build_user account
mkdir buildAgent/logs  # Directory should be created under your_build_user user
sh buildAgent/bin/mac.launchd.sh load
```

  You have to wait several minutes for the build agent to auto-upgrade from the TeamCity server. You can watch the process in the logs:

```
tail -f buildAgent/logs/teamcity-agent.log
```

- When the build agent is upgraded and successfully connects to TeamCity server, stop it:

```
sh buildAgent/bin/mac.launchd.sh unload
```

- After buildAgent upgrade from the TeamCity server, copy the `buildAgent/bin/jetbrains.teamcity.BuildAgent.plist` file to `$HOME/Library/LaunchAgents/` directory.
- Configure your Mac system to automatically login as a build user, as described here
- Reboot

On the system startup, the build user should automatically log in, and the build agent should start.

Configure a second build agent on macOS

If you want to start several build agents, repeat the procedure of installing and starting build agent with the following changes:

- Install the second build agent in a different directory
- In `conf/buildAgent.properties`, you should specify a different name and port
- In the `$HOME/start_build_agent.command` (see details above), add a line to start the second build agent

Stopping the Build Agent

To stop the agent manually, run the `<Agent home>\agent` script with a `stop` parameter.

Use `stop` to request stopping after the current build finished.
Use `stop force` to request an immediate stop (if a build is running on the agent, it will be stopped abruptly (canceled))
Under Linux, you have one more option top use: `stop kill` to kill the agent process.

If the agent runs with a console attached, you may also press Ctrl+C in the console to stop the agent (if a build is running, it will be canceled).

Configuring Java

A TeamCity build agent is a Java application and it requires JDK version 6-10 to work. OpenJDK 8 (e.g. by AdoptOpenJDK) 1.8.0_161 or later, 32-bit is recommended. Oracle Java 8 is also supported.

A build agent contains two processes:

- Agent Launcher — a Java process that launches the agent process
- Agent — the main process for a Build Agent; runs as a child process for the agent launcher

The (Windows) .exe TeamCity distribution comes bundled with Java 1.8.0_161.
If you run a previous version of the TeamCity agent, you will need to repeat the agent installation to update the JVM.

Using x32 bit JDK (not JRE) is recommended. JDK is required for some build runners like IntelliJ IDEA Project, Java Inspections and Duplicates. If you do not have Java builds, you can install JRE instead of JDK.
Using of x64 bit Java is possible, but you might need to double the -Xmx memory value for the main agent process (see Configuring Build Agent Startup Properties and alike section for the server).

For the .zip agent installation you need to install the appropriate Java version. Make it available via PATH or available in one of the following places:

- the `<Agent home>/jre` directory
- in the directory pointed to by the `TEAMCITY_JRE`, `JAVA_HOME` or `JRE_HOME` environment variables (check that you only have one of the variables defined).
- if you plan to run the agent via Windows service, make sure to set `wrapper.java.command` property in the `<agent home>\launcher\conf\wrapper.conf` file to a valid path to the Java executable

Upgrading Java on Agents

If a build agent uses a Java version older than the one required by agent (Java 6 currently), the agent will not be able to start and will be shown as disconnected.

If a build agent uses a Java version older than Java 8 (e.g. Java 6 or 7), you will see the corresponding warning on the agent's page and a health item in the web UI.

> ⚠ Support for Java prior to version 8 will be dropped in future TeamCity versions. Consider upgrading Java on the agent if you see the warning.

It is recommended to use latest Java 8, 32 bit version. OpenJDK 8 (e.g. by AdoptOpenJDK) 1.8.0_161 or later, 32-bit is recommended. Oracle Java 8 is also supported.

To update Java on agents, do one of the following:

- Switch to using newer Java: if the appropriate Java version of the same bitness as the current one is detected on the agent, the agent page provides an action to switch to using that Java automatically. Upon the action invocation, the agent process is restarted (once the agent becomes idle, i.e. finishes the current build if there is one) using the new Java.
- (Windows) Since the build agent `.exe` installation comes bundled with the required Java, you can just manually reinstall the agent using the `.exe` installer obtained from the TeamCity server | Agents page.
- Install a required Java on the agent into one of the standard locations, and restart the agent - the agent should then detect it and provide an action to use a newer Java in the web UI (see above).
- Install a required Java on the agent and configure the agent to use it.

> ⚠ In a rare case of updating the Java for the process that launches the TeamCity agent, use one of the options for the agent Java upgrade.
> Another way for Build Agent started as a Windows service, is to stop the service, change the `wrapper.java.command` variable in `buildAgent\launcher\conf\wrapper.conf` to point to the new `java.exe` binary, and restart the service.

Installing Several Build Agents on the Same Machine

You can install several TeamCity agents on the same machine if the machine is capable of running several builds at the same time. However, we recommend running a single agent per (virtual) machine to minimize builds cross-influence and making builds more predictable. When installing several agents, it is recommended to install them under different OS users so that user-level resources (like Maven/Gradle/NuGet local artifact caches) do not conflict.

TeamCity treats all agents equally regardless of whether they are installed on the same or on different machines.
When installing several TeamCity build agents on the same machine, please consider the following:

- The builds running on such agents should not conflict by any resource (common disk directories, OS processes, OS temp directories).
- Depending on the hardware and the builds, you may experience degraded builds' performance. Ensure there are no disk, memory, or CPU bottlenecks when several builds are run at the same time.
- It is recommended to set up the agents to be run under different OS users

After having one agent installed, you can install additional agents by following the regular installation procedure (see an exception for the Windows service below), but make sure that:

- The agents are installed in separate directories.
- The agents have the distinctive `workDir` and `tempDir` directories in the `buildAgent.properties` file.
- Values for the `name` and `ownPort` properties of `buildAgent.properties` are unique.
- No builds running on the agents have the absolute checkout directory specified.

Make sure there are no build configurations with the absolute checkout directory specified (alternatively, make sure such build configurations have the "clean checkout" option enabled and they cannot be run in parallel).

Usually, for a new agent installation you can just copy the directory of the existing agent to a new place with the exception of its "temp", "work", "logs" and "system" directories. Then, modify `conf/buildAgent.properties` with the new `name`, `ownPort` values. Please also clear (delete or remove the value) for the `authorizationToken` property and make sure the `workDir` and `tempDir` are relative/do not clash with another agent.

If you use Windows installer to install additional agents and want to run the agent as a service, you will need to perform manual steps as installing second agent as a service on the same machine is not supported by the installer: the existing service is overwritten (see also a feature request).
In order to install the second agent, it is recommended to install the second agent manually (using .zip agent distribution). You

can use Windows agent installer and do not opt for service installation, but you will lose uninstall option for the initially installed agent this way.
After the second agent is installed, register a new service for it as mentioned in the section above.

> For step-by-step instructions on installing a second Windows agent as a service, see a related external blog post.

See also:

- Concepts: Build Agent