Setting Up TeamCity for Amazon EC2

TeamCity Amazon EC2 integration allows you to configure TeamCity with your Amazon account and then start and stop images with TeamCity agents on-demand based on the queued builds.

For integrations with other cloud solutions, see the following pages:

- VMWare vSphere (bundled)
- Windows Azure
- Google Cloud Agents
- Implementing Cloud support enables to you create your own integration.

On this page:

General Description

It is assumed that the machine images are pre-configured to start TeamCity agent on boot (see details below). The exception is the usage of agent push.

Once a cloud profile is configured in TeamCity with one or several images, TeamCity does a test start for all the new images to learn about the agents on them.
Once the agents are connected, TeamCity stores their parameters to be able to process the build configurations-to-agents compatibility correctly.

For each queued build, TeamCity first tries to start it on one of the regular, non-cloud agents. If there are no usual agents available, TeamCity finds a matching cloud image with a compatible agent and starts a new instance for the image. TeamCity ensures that the running instances limit configured in the cloud profile is not exceeded.

Once an agent is connected from a cloud instance started by TeamCity, it is automatically authorized (provided there are available agent licenses). After that, the agent is processed as a regular agent.
If running timeout is configured on the cloud profile and it is reached, the instance is terminated.
If an EBS-based instance id is specified in the images list, the instance is stopped instead.

On instance terminating/stopping, its disconnected agent is removed from authorized agents list and is deleted from the system.

Amazon EC2 Spot Instances are supported.

Configuration

Understanding Amazon EC2 and ability to perform EC2 tasks is a prerequisite for configuring and using TeamCity Amazon EC2 integration.

Basic TeamCity EC2 setup involves:

- preparing an Amazon EC2 image (AMI) with an installed TeamCity agent
- configuring EC2 integration on TeamCity server

⚠ Please note that the number of EC2 agents is limited by the total number of agent licenses you have in TeamCity.

Please ensure that the server URL specified on the Global Settings page in the Administration area is correct since agents will use it to connect to the server.

If you need TeamCity to use a proxy to access EC2 services, please read on a current workaround in the dedicated issue.

Required IAM permissions

TeamCity requires the following permissions for Amazon EC2 Resources:

- `ec2:Describe*`
- `ec2:StartInstances`
- `ec2:StopInstances`
- `ec2:TerminateInstances`
- `ec2:RebootInstances`
- `ec2:RunInstances`
- `ec2:ModifyInstanceAttribute`
- `ec2:Tags`

To use spot instances, the following additional permissions are required:

- `ec2:RequestSpotInstances`
- `ec2:CancelSpotInstanceRequests`

To launch an instance with the Iam Role (applicable to instances cloned from AMI-s only), the following additional permissions are required:

- `iam:ListInstanceProfiles`
- `iam:PassRole`

`An example of custom IAM policy definition (allows all ec2 operations from a specified IP address):`

AWS IAM Policy Definition Example

› Expand source

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "1",
         "Effect": "Allow",
         "Action": "ec2:*",
         "Condition": {
            "IpAddress": {
               "aws:SourceIp": "<TeamCity server IP address>"
            }
         },
         "Resource": "*"
      }
   ]
}
```

Optional permissions

See the section below for permissions to set IAM roles on an agent instance.

View information on example policies for Linux and Windows on the Amazon website.

Preparing Image with Installed TeamCity Agent

Here are the requirements for an image that can be used for TeamCity cloud integration:

- TeamCity agent should be correctly installed.
- TeamCity agent should start on machine startup
- `buildAgent.properties` can be left as is. The "`serverUrl`, "`name`" and "`authorizationToken`" properties can be empty or set to any value; they are ignored when TeamCity starts the instance.

Provided these requirements are met, usual TeamCity agent installation and cloud-provider image bundling procedures are applicable.

If you need the connection between the server and the agent machine to be secure, you will need to set up the agent machine to establish a secure tunnel (e.g., VPN) to the server on boot so that TeamCity agent receives data via the secure channel.

Recommended image (e.g., Amazon AMI) preparation steps:

1. Choose one of the existing generic images.
2. Start the image.
3. Configure the running instance:
    - Install and configure a build agent:
        - Configure server name and agent name in `conf/buildAgent.properties` — this is optional if the image will be started by TeamCity, but it is useful to test the agent is configured correctly.
        - It usually makes sense to specify `tempDir` and `workDir` in `conf/buildAgent.properties` to use non-system drive (`d:` under Windows)
    - Install any additional software necessary for the builds on the machine.
    - Run the agent and check it is working OK and is compatible with all necessary build configurations, etc.
    - Configure system so that agent it is started on machine boot (and make sure TeamCity server is accessible on machine boot).
    - To ensure proper TeamCity agent communication with EC2 API (including access to additional drives) on Windows, add a dependency from the TeamCity Build Agent service on the AmazonSSMAgent or EC2Launch / EC2Config service (the service which makes sure the machine is fully initialized in regard to AWS infrastructure use). This can be done, for example, via the Registry or using sc config, e.g. `sc config TCBuildAgent depend=EC2Config`
      Alternatively, you can use the "Automatic (delayed start)" service starting mode.
4. Test the setup by rebooting the machine and checking that the agent connects normally to the server.
5. Prepare the Image for bundling:
    - Remove any temporary/history information in the system.
    - Stop the agent (under Windows stop the service but leave it in Automatic startup type)
    - Delete content `logs` and `temp` directories in agent home (optional)
    - Delete "`<Agent Home>/conf/amazon-*`" file (optional)
    - Change `config/buildAgent.properties` to remove properties: `name`, `serverUrl`, `authorizationToken` (optional). `serverUrl` can be removed for EC2 integration plugins. Other plugins might require that it is present and set to correct value.
6. Make a new image from the running instance (or just stop it for Amazon EBS images).

Additional configuration for Windows agents

To ensure proper TeamCity agent communication with EC2 API (including access to additional drives) on Windows, add a dependency from the TeamCity Build Agent service on the AmazonSSMAgent or EC2Launch / EC2Config service (the service which makes sure the machine is fully initialized in regard to AWS infrastructure use). This can be done, for example, via the Registry or using sc config, e.g., `sc config TCBuildAgent depend=EC2Config`
Alternatively, you can use the "Automatic (delayed start)" service starting mode.

Important note for images based on Windows Server 2016 image

Due to the bug in the network settings, instance meta-data is not available by default. That means TeamCity agent service cannot retrieve its properties and cloud integration doesn't work (agent does not connect to the server or is not automatically authorized).
To fix the issue, do the following:
1) Install the latest EC2Config
2) When EC2Config is installed make TCBuildAgent service dependent on both EC2Config and AmazonSSMAgent via the command: `sc config TCBuildAgent depend=Ec2Config/AmazonSSMAgent`

Agent auto-upgrade Note

TeamCity agent auto-upgrades whenever distribution of agent plugins on the server changes (e.g., after TeamCity upgrade). If you want to cut agent startup time, you might want to re-bundle the agent AMI after agent plugins have been auto-updated.

Configuring a cloud profile in TeamCity

Next configure Amazon EC2 Agent Cloud Profile in the Server Administration UI, on the Administration | Agent Cloud.

IAM profiles

It is possible to use IAM profiles with build agents launched as Amazon EC2 instances, which requires the supplied AWS account to have the following permissions:

- `iam:ListInstanceProfiles`
- `iam:PassRole`

IAM profiles must be preconfigured in Amazon EC2. In the TeamCity Web UI, the IAM profile dropdown enables you to select a role. Every new launched EC2 instance will assume the selected IAM role.

Amazon EC2 Spot Instances support

TeamCity supports Amazon EC2 Spot Instances and now you can place your bid on unused EC2 capacity and use it, as long as your suggested price exceeds the current "Spot price".

Enable spot instances on the TeamCity cloud profile page and enter your current bid level.
After the profile is saved, TeamCity creates a spot instance request (sir) and the availability of Amazon spot instances is checked. If the sir is not fulfilled after 10 min, it is killed by Amazon.
If there are still queued builds which can run on such an agent, the request is automatically recreated by TeamCity.
NOTE: It is not recommended to use spot instances for production-critical builds due to the possibility of an unexpected spot instance termination by Amazon.

Amazon EBS-Optimized Instances

The behavior of EBS-optimization in TeamCity 10.0 is similar to that offered by EC2 console. When configuring an image of the Amazon cloud profile, the optimization can be set using the corresponding box of the Instance Type.
Note that

- EBS-optimization is turned on by default for `c4.*` , `d2.*`, and `m4.*` (non-configurable)
- EBS-optimization is turned off by default for any other instance types and can be turned on for instances that support it (such as `c3.xlarge`, etc.)

Tagging for TeamCity-launched instances

Requirements

The following requirements must be met for tagging instances launched by TeamCity:

- you have the `ec2:*Tags` permissions
- the maximum number of tags (50) for your Amazon EC2 resource is not reached.

In the absence of tagging permissions, TeamCity will still launch Amazon AMI and EBS images with no tags applied; Amazon EC2 Spot Instances will not be launched.

Automatic tags

TeamCity enables users to get instance launch information by marking the created instances with the `"teamcity:TeamcityData"` tag containing `<server UUID>:-<cloud profile ID>:-<image reference>`.

Custom tags

Custom tags can be applied to EC2 cloud agent instances: when configuring Cloud profile settings, in the Add Image/ Edit Image dialog use the Instance tags: field to specify tags in the format of `<key1>=<value1>,<key2>=<value2>`. Amazon tag restrictions need to be considered.

If you'd like to use equal(=) sign in the tag value, no escaping is needed. For instance, the string `extraParam=name=John` will be parsed into `<key=extraParam>` and value `<name=John>`.

Tagging instance-dependent resources

When launching Amazon EC2 instances, TeamCity tags all the resources (e.g. volumes and network adapters) associated with the created instances, which is important when evaluating the overall cost of an instance (taking into account the storage drive type and size, I/O operations (for standard drives), network (transfers out), etc.

Sharing single EBS instance between several TeamCity servers

As mentioned above, TeamCity tags every instance it launches with the `teamcity:TeamcityData` tag that represents a server, cloud profile, and source (AMI or EBS-instance). So, when several TeamCity servers try to use the same EBS instance, the second one will see the following message "Instance is used by another TeamCity server. Unable to start/stop it". If you are sure that no other TeamCity servers are working with this instance, you can delete the "`teamcity:TeamcityData`" tag and the instance will become available for all TeamCity servers again.

New instance types

Since Amazon doesn't provide a robust API method to retrieve all instance types, Amazon integration relies on the periodical update of AWS SDK to make new instance types available.

However, there is a workaround if you are not willing to wait. To register new Instance Types, use the following internal property:

`teamcity.ec2.instance.types` property with new instance types separated by ","

Proxy settings

If your TeamCity server needs to use a proxy to connect to AWS API endpoint, configure the following server internal properties to connect to Amazon AWS addresses.

`teamcity.http.proxy.host.ec2` - proxy server host name
`teamcity.http.proxy.port.ec2` - proxy server port

For proxy server authentication:
`teamcity.http.proxy.user.ec2` - proxy access user name
`teamcity.http.proxy.password.ec2` - proxy access user password

For NTML authentication:
`teamcity.http.proxy.domain.ec2` - proxy user domain for NTLM authentication
`teamcity.http.proxy.workstation.ec2` - proxy access workstation for NTLM authentication

Custom script

It is possible to run a custom script on the instance start (applicable to instances cloned from AMI's only). The Amazon website details the script format for Linux and Windows.

Estimating EC2 Costs

Usual Amazon EC2 pricing applies. Please note that Amazon charges can depend on the specific configuration implemented to deploy TeamCity. We advice you to check your configuration and Amazon account data regularly in order to discover and prevent unexpected charges as early as possible.

Please note that traffic volumes and necessary server and agent machines characteristics depend a big deal on the TeamCity setup and nature of the builds run. See also How To...#Estimate Hardware Requirements for TeamCity.

Traffic Estimate

Here are some points to help you estimate TeamCity-related traffic:

- If TeamCity server is not located within the same EC2 region or availability zone that is configured in TeamCity EC2 settings for agents, traffic between the server and agent is subject to usual Amazon EC2 external traffic charges.
- When estimating traffic please bear in mind that there are lots types of traffic related to TeamCity (see a non-complete list below).

External connections originated by server:

- VCS servers
- Email and Jabber servers
- Maven repositories

Internal connections originated by server:

- TeamCity agents (checking status, sending commands, retrieving information like thread dumps, etc.)

External connections originated by agent:

- VCS servers (in case of agent-side checkout)
- Maven repositories
- any connections performed from the build process itself

Internal connections originated by agent:

- TeamCity server (retrieving build sources in case of server-side checkout or personal builds, downloading artifacts, etc.)

Usual connections served by the server:

- web browsers
- IDE plugins

Uptime Costs

As Amazon rounds machine uptime to the nearest full hour, please adjust timeout setting on the EC2 image setting on TeamCity cloud integration settings according to your usual builds length.

It is also highly recommended to set execution timeout for all your builds so that a build hanging does not cause prolonged instance running with no payload.