# Exercise - Create a build agent that runs on Azure

This module requires a sandbox to complete. A [**sandbox**](https://docs.microsoft.com/en-us/learn/support/faq?pivots=sandbox) gives you access to Azure resources. Your Azure subscription will not be charged. The sandbox may only be used to complete training on Microsoft Learn. Use for any other reason is prohibited, and may result in permanent loss of access to the sandbox.

Due to the impact of the global health pandemic, Azure resources are being prioritized towards health and safety organizations. You may experience some issues when you deploy resources used in the exercises. Please try again or choose a different region. For more information, see Azure blog post - [**Update #3: Business continuity with Azure**](https://azure.microsoft.com/blog/update-3-business-continuity-azure/).

**Activate sandbox**

In this unit, to configure a build agent that you can use in Microsoft Azure Pipelines, you use a virtual machine that runs on Microsoft Azure. We provide a virtual machine that you can use for the duration of this module.

In this unit, you will:

* Create an Ubuntu virtual machine on Azure to serve as your build agent.
* Create an agent pool in Microsoft Azure DevOps.
* Create an access token to authenticate your agent with Azure DevOps.
* Configure your agent with the software that's required to build the Space Game website.
* Configure your agent to connect to Azure DevOps so that it can receive build jobs.
* Verify that the agent is connected to Azure DevOps and ready to receive build jobs.

There are many ways to create a virtual machine on Azure. In this unit, you create an Ubuntu virtual machine by using an interactive terminal called Cloud Shell.

To configure your VM, you have several choices:

* For a Linux VM, you can connect directly over SSH and interactively configure your system.
* You can automate the deployment by using an Azure Resource Manager template.
* If you need to deploy many build agents, you can create a VM image that has all the software pre-installed.

Configuring a system interactively is a good way to get started, because it helps you understand the process and what's needed. To simplify the process, connect to your Ubuntu VM over SSH and run shell scripts to set up your build agent.

**Note**

If you're unfamiliar with connecting to or configuring Linux systems, just follow along. You can apply the same concepts to Windows build agents.

## Create a Linux virtual machine

In this section, you create a VM that's running Ubuntu 18.04, which will serve as your build agent. The VM isn't yet set up to be a build agent or have any of the tools that are required to build the Space Game web application. You'll set that up shortly.

To create your VM, in Cloud Shell (at right), run the following az vm create command:

**Azure CLI**

az vm create \

--name MyLinuxAgent \

--resource-group [Resource Group Name] \

--image Canonical:UbuntuServer:18.04-LTS:latest \

--location eastus \

--size Standard\_DS2\_v2 \

--admin-username azureuser \

--generate-ssh-keys

Your VM will take a few minutes to come up.

[Standard\_DS2\_v2](https://docs.microsoft.com/en-us/azure/virtual-machines/dv2-dsv2-series#dsv2-series) specifies the VM's size. A VM's size defines its processor speed, amount of memory, initial amount of storage, and expected network bandwidth. This is the same size that's provided by Microsoft-hosted agents. In practice, you can choose a size that provides more compute power or additional capabilities, such as graphics processing.

The --resource-group argument specifies the resource group that holds all the things that we need to create. A resource group enables you to administer all the VMs, disks, network interfaces, and other elements that make up our solution as a unit. Normally, you would create your own resource group before you create Azure resources. Because you're in the free Azure sandbox environment, you can skip this step. Instead, you use the pre-created resource group [Resource Group Name].

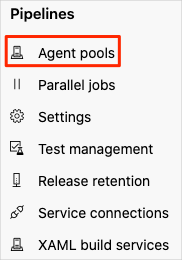
**Important**

The Azure sandbox gives you temporary access to Azure resources. When your session expires, the VM you create here will no longer be accessible to you as a build agent. In practice, you would set up a build agent by using your own Azure subscription or a system that's running in your datacenter.

## Create the agent pool

Recall that an agent pool organizes build agents. In this section, you create the agent pool in Azure DevOps. Later, you'll specify the name of the agent pool when you configure your agent so that it can register itself to the correct pool.

1. In Azure DevOps, go to the **Space Game - web - Agent** project.
2. Select **Project settings**.
3. Under **Pipelines**, select **Agent pools**.



1. Select **Add pool**.
2. In the **Add pool** window:
   1. Under **Pool to link**, select **New**.
   2. Under **Pool type**, select **Self-hosted**.
   3. Under **Name**, enter MyAgentPool.

In practice, you would choose a more descriptive name for the purpose of your pool.

1. Select **Create**. The new agent pool appears in the list.

## Create a personal access token

For your build agent to register itself with Azure DevOps, you need a way for it to authenticate itself.

To do that, you create a personal access token. A personal access token, or PAT, is an alternative to a password. You can use the PAT to authenticate with services such as Azure DevOps.

**Important**

As you would with a password, be sure to keep your access token in a safe place. In this section, you store your access token as an environment variable so that it doesn't appear in your shell script.

1. In Azure DevOps, open your profile settings, and then select **Personal access token**.

Locating SecurityPersonal Access Token in the menu

1. Select **New Token**.
2. Enter a name for your token, such as Build agent.
3. Under **Scopes**, select **Show all scopes** at the bottom.
4. Look for **Agent Pools**, and then select **Read & manage**.
5. Look for **Build**, and then select **Read & execute**.
6. Select **Create**.
7. Copy the token to a safe place.

Shortly, you'll use your token to enable your build agent to authenticate access to Azure Pipelines.

### Connect to your VM

In this section, you connect to your Linux VM over SSH so that you can configure it.

Recall that you can't interactively sign in to a Microsoft-hosted agent. Because a private build agent is your own, you can sign in to and configure it however you'd like.

The ability to connect to your build agent enables you to configure it with the tools you need to build your software. It also enables you to troubleshoot issues as you build out your pipeline configuration.

1. To get your VM's IP address, in Cloud Shell, run az vm show:

**Azure CLI**

IPADDRESS=$(az vm show \

--name MyLinuxAgent \

--resource-group [Resource Group Name] \

--show-details \

--query [publicIps] \

--output tsv)

This command stores the IP address in a Bash variable named IPADDRESS.

1. Print the VM's IP address to the console.

**Bash**

echo $IPADDRESS

1. Create an SSH connection to your VM. At the prompt, enter **yes** to continue connecting.

**Bash**

ssh azureuser@$IPADDRESS

You're now connected to your VM over SSH.

This command works because you provided the --generate-ssh-keys option when you ran az vm create earlier. This option creates an SSH key pair, which enables you to sign in to the VM.

## Install build tools on your VM

In this section, you configure your VM with the tools that are required to build the Space Game website.

Recall that your existing build process uses these tools:

* .NET Core
* Node.js, which is used to perform build tasks
* npm, the package manager for Node.js
* gulp, a Node.js package that's used to minify JavaScript and CSS files

These are the primary tools that the build process requires. To install them, you download and run a shell script from GitHub.

**Note**

The build process uses other tools, such as node-sass, to convert Sass (.scss) files to CSS (.css) files. However, Node.js installs these tools when the build runs.

Let's start by updating the Ubuntu package manager, named apt. This action fetches the latest information from the package repositories and is ordinarily the first thing you do when you set up a new Ubuntu system.

1. In your SSH connection, update the apt package manager cache.

**Bash**

sudo apt-get update

sudo runs the command with administrator, or root, privileges.

1. To download a shell script named build-tools.sh from GitHub, run the following curl command:

**Bash**

curl https://raw.githubusercontent.com/MicrosoftDocs/mslearn-azure-pipelines-build-agent/master/build-tools.sh > build-tools.sh

1. Print the script to the terminal so that you can examine its contents.

**Bash**

cat build-tools.sh

The following result is displayed:

**Bash**

#!/bin/bash

set -e

# Select a default .NET version if one is not specified

if [ -z "$DOTNET\_VERSION" ]; then

DOTNET\_VERSION=3.1.300

fi

# Add the Node.js PPA so that we can install the latest version

curl -sL https://deb.nodesource.com/setup\_14.x | bash -

# Install Node.js, npm, and jq

apt-get install -y nodejs npm jq

# Install gulp

npm install -g gulp

# Change ownership of the .npm directory to the sudo (non-root) user

chown -R $SUDO\_USER ~/.npm

# Install .NET as the sudo (non-root) user

sudo -i -u $SUDO\_USER bash << EOF

curl -sSL https://dot.net/v1/dotnet-install.sh | bash /dev/stdin -c Current -v $DOTNET\_VERSION

EOF

The script installs Node.js, npm, gulp, and .NET Core.

By setting the DOTNET\_VERSION environment variable, you can specify the .NET version to install. If you don't set this variable, the script installs the version that your existing build configuration uses. For learning purposes, you don't set this variable. You allow the script to use the default version.

1. Make the script executable, and then run the script.

**Bash**

chmod u+x build-tools.sh

sudo ./build-tools.sh

The script takes a few minutes to run.

In practice, you could now run commands to verify that each software component was successfully installed.

## Install agent software on your VM

Now it's time to install the agent software on your VM. This software enables the VM to act as a build agent and receive build jobs from Azure Pipelines.

The registration process checks for installed software before it registers the agent with Azure Pipelines. Therefore, it's important to set up the agent after you install all other software. In practice, you can register the agent a second time if you need to install additional software.

The documentation explains how to manually set up [self-hosted Linux agents](https://docs.microsoft.com/en-us/azure/devops/pipelines/agents/v2-linux?view=azure-devops) as well as macOS and Windows agents. You run a shell script to configure your agent, much like the way you set up build tools in the preceding section.

**Important**

The script that you run here is for learning purposes. In practice, you should first understand how each command in the scripts you build affect the overall system. At the end of the module, we'll point to documentation that more completely describes your options.

1. To download a shell script named build-agent.sh from GitHub, run the following curl command:

**Bash**

curl https://raw.githubusercontent.com/MicrosoftDocs/mslearn-azure-pipelines-build-agent/master/build-agent.sh > build-agent.sh

1. Print the script to the terminal so that you can examine its contents.

**Bash**

cat build-agent.sh

The following result is displayed:

**Bash**

#!/bin/bash

set -e

# Select a default agent version if one is not specified

if [ -z "$AZP\_AGENT\_VERSION" ]; then

AZP\_AGENT\_VERSION=2.164.1

fi

# Verify Azure Pipelines token is set

if [ -z "$AZP\_TOKEN" ]; then

echo 1>&2 "error: missing AZP\_TOKEN environment variable"

exit 1

fi

# Verify Azure DevOps URL is set

if [ -z "$AZP\_URL" ]; then

echo 1>&2 "error: missing AZP\_URL environment variable"

exit 1

fi

# If a working directory was specified, create that directory

if [ -n "$AZP\_WORK" ]; then

mkdir -p "$AZP\_WORK"

fi

# Create the Downloads directory under the user's home directory

if [ -n "$HOME/Downloads" ]; then

mkdir -p "$HOME/Downloads"

fi

# Download the agent package

curl https://vstsagentpackage.azureedge.net/agent/$AZP\_AGENT\_VERSION/vsts-agent-linux-x64-$AZP\_AGENT\_VERSION.tar.gz > $HOME/Downloads/vsts-agent-linux-x64-$AZP\_AGENT\_VERSION.tar.gz

# Create the working directory for the agent service to run jobs under

if [ -n "$AZP\_WORK" ]; then

mkdir -p "$AZP\_WORK"

fi

# Create a working directory to extract the agent package to

mkdir -p $HOME/azp/agent

# Move to the working directory

cd $HOME/azp/agent

# Extract the agent package to the working directory

tar zxvf $HOME/Downloads/vsts-agent-linux-x64-$AZP\_AGENT\_VERSION.tar.gz

# Install the agent software

./bin/installdependencies.sh

# Configure the agent as the sudo (non-root) user

chown $SUDO\_USER $HOME/azp/agent

sudo -u $SUDO\_USER ./config.sh --unattended \

--agent "${AZP\_AGENT\_NAME:-$(hostname)}" \

--url "$AZP\_URL" \

--auth PAT \

--token "$AZP\_TOKEN" \

--pool "${AZP\_POOL:-Default}" \

--work "${AZP\_WORK:-\_work}" \

--replace \

--acceptTeeEula

# Install and start the agent service

./svc.sh install

./svc.sh start

You don't need to understand how each line works, but here's a brief summary of what this script does:

* + It downloads the agent package as a .tar.gz file and extracts its contents.
  + In the extracted files, the script:
    - Runs a shell script named installdependencies.sh to install the agent software.
    - Runs a shell script named config.sh to configure the agent and register the agent with Azure Pipelines.
    - Runs a shell script named svc.sh to install and start the agent service.

The script uses environment variables to enable you to provide details about your Azure DevOps organization. Here's a summary:

| **TABLE 1** | | |
| --- | --- | --- |
| **Bash variable** | **Description** | **Default** |
| AZP\_AGENT\_VERSION | The [version of the agent software](https://github.com/Microsoft/azure-pipelines-agent/releases) to install | The version we last used to test this module |
| AZP\_URL | The URL of your Azure DevOps organization | (None) |
| AZP\_TOKEN | Your personal access token | (None) |
| AZP\_AGENT\_NAME | Your agent's name as it appears in Azure DevOps | The system's hostname |
| AZP\_POOL | The name of your agent pool | **Default** |
| AZP\_WORK | The working directory for the agent to perform build tasks | **\_work** |

If the script doesn't provide a default value for a variable that's not set, the script prints an error message and immediately exits.

In the steps that follow, set these environment variables:

* + AZP\_AGENT\_VERSION
  + AZP\_URL
  + AZP\_TOKEN
  + AZP\_AGENT\_NAME
  + AZP\_POOL

For now, we recommend that you leave the other variables unset.

1. Set the AZP\_AGENT\_NAME environment variable to specify your agent's name. We recommend **MyLinuxAgent**.

**Bash**

export AZP\_AGENT\_NAME=MyLinuxAgent

1. Set the AZP\_URL environment variable to specify the URL to your Azure DevOps organization.

Replace <organization> with yours. You can get the name from the browser tab that displays Azure DevOps.

**Bash**

export AZP\_URL=https://dev.azure.com/organization

1. Set the AZP\_TOKEN environment variable to specify your personal access token.

Replace <token> with your token.

**Bash**

export AZP\_TOKEN=token

1. Set the AZP\_POOL environment variable to specify the name of your agent pool. Earlier, you created a pool named MyAgentPool.

**Bash**

export AZP\_POOL=MyAgentPool

1. Set the AZP\_AGENT\_VERSION environment variable to specify the latest version of the agent.

**Bash**

export AZP\_AGENT\_VERSION=$(curl -s https://api.github.com/repos/microsoft/azure-pipelines-agent/releases | jq -r '.[0].tag\_name' | cut -d "v" -f 2)

A YAML pipeline on a Linux machine must be using the latest version of the agent, even if it is pre-release. The agent software is constantly updating, so you curl the version information from [the GitHub repo](https://api.github.com/repos/microsoft/azure-pipelines-agent/releases). The command uses jq to read the latest version from the JSON string that's returned.

1. Print the agent version to the console. Optionally, [check](https://github.com/microsoft/azure-pipelines-agent/releases) to make sure this is the latest version.

**Bash**

echo $AZP\_AGENT\_VERSION

1. Make the script executable, and then run it.

**Bash**

chmod u+x build-agent.sh

sudo -E ./build-agent.sh

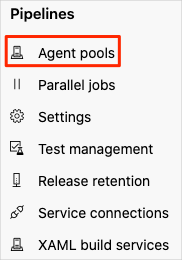
sudo enables the script to run as the root user. The -E argument preserves the current environment variables, including the ones you set, so that they are available to the script.

As the script runs, you can see the agent connect to Azure DevOps, see it added to the agent pool, and see the agent connection be tested.

## Verify that the agent is running

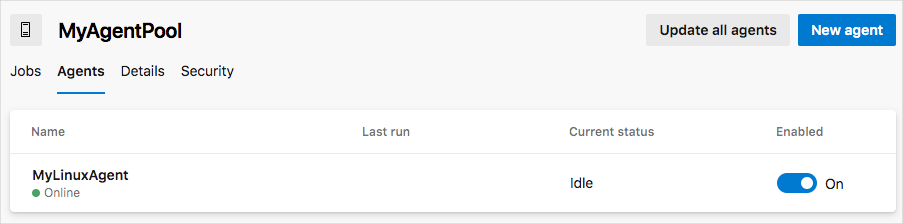
You've successfully installed the build tools and the agent software on your VM. As a verification step, go to Azure DevOps and see your agent in the agent pool.

1. In Azure DevOps, go to the **Space Game - web - Agent** project.
2. Select **Project settings**.
3. Under **Pipelines**, select **Agent pools**.



1. Select **MyAgentPool**.
2. Select the **Agents** tab.

You can see that your agent is online and ready to accept build jobs.



**Tip**

If your build agent shows as **Offline**, try waiting a few moments and then refreshing the page.

1. Select your agent, **MyLinuxAgent**.
2. Select the **Capabilities** tab.

During setup, the configuration process scanned your build agent for tool capabilities. You see that npm is listed as one of them. Recall that your original build configuration specified that npm must be installed on the agent.



When you specify which agent pool to use, you can include any of these entries in your demands section. Including them ensures that Azure Pipelines chooses a build agent that has the software you need to build your application. It also enables you to create agent pools with various software configurations. Azure Pipelines will select the correct configuration based on your requirements.