100 XP

Exercise - Run Terraform in Visual Studio Code

6 minutes

You can also run Terraform configuration files using Visual Studio Code. It uses other Terraform services that you can integrate with Visual Studio Code.

Two Visual Studio code extensions that are required are **Azure Account**, and **Terraform**.

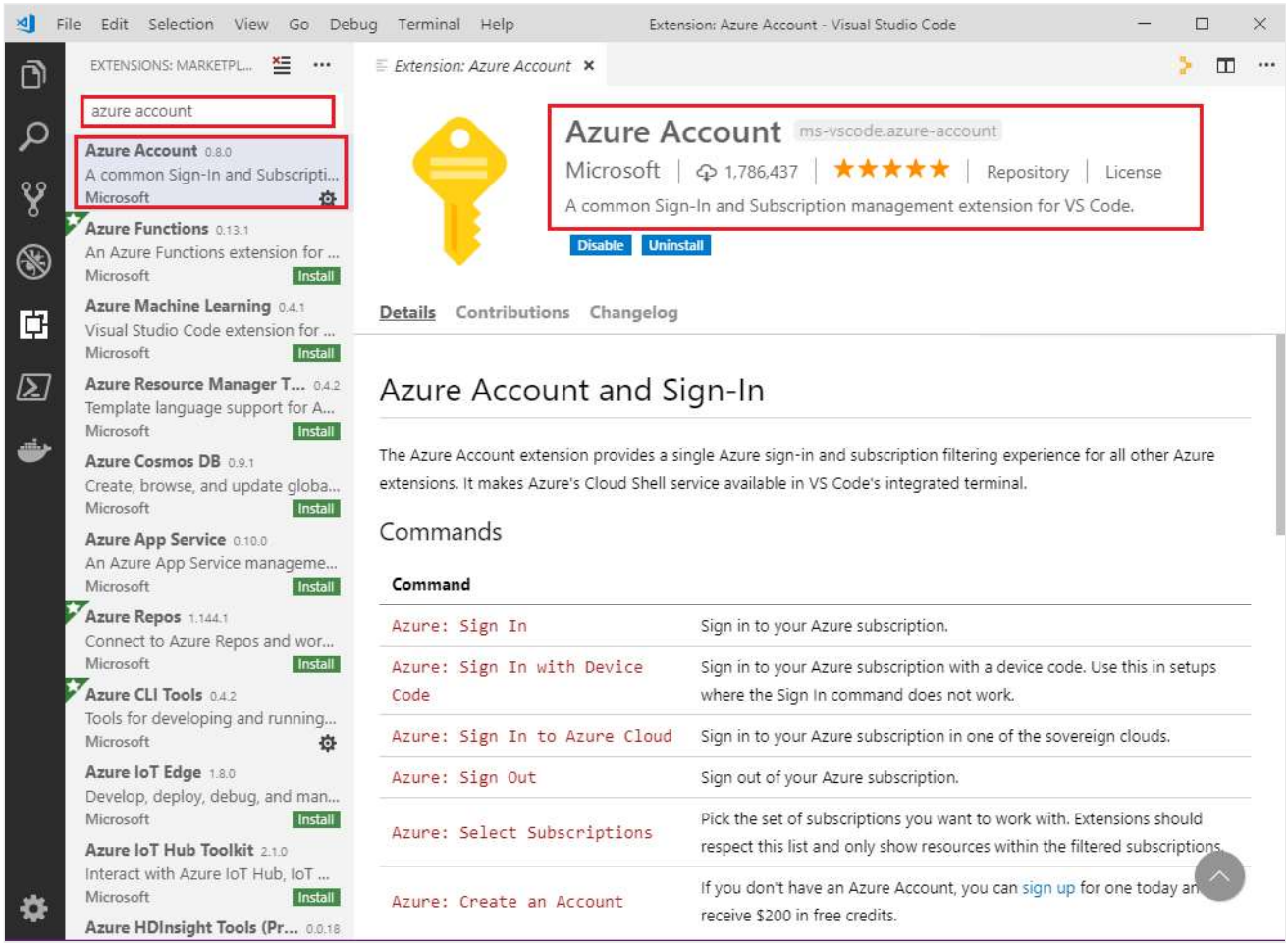
In this walkthrough, you'll create a VM in Visual Studio Code using Terraform.

Prerequisites

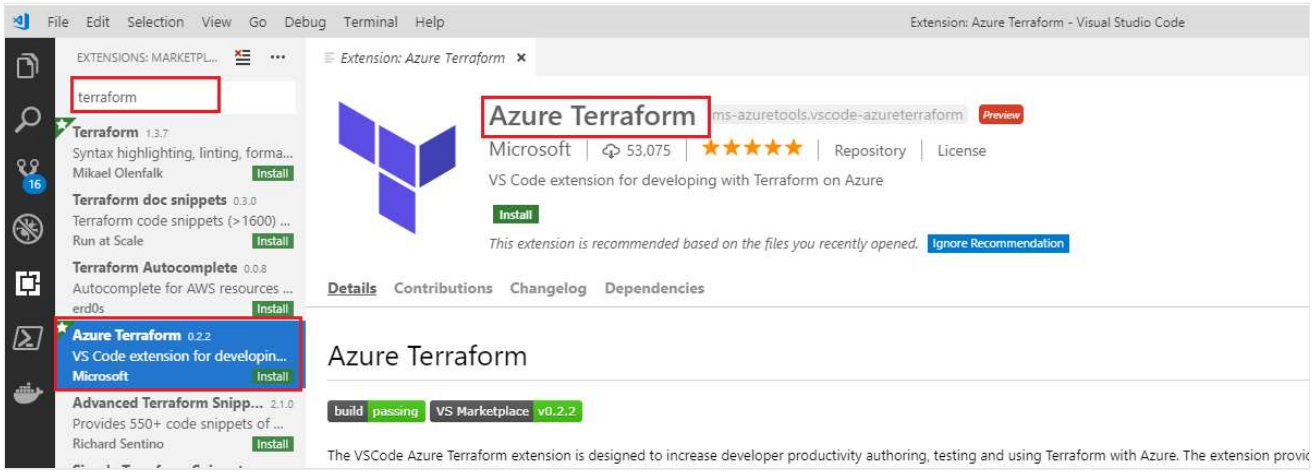
- This walkthrough requires Visual Studio Code. If you don't have Visual Studio Code installed, you can download it from https://code.visualstudio.com/ . Download and install a version of Visual Studio Code appropriate to your operating system environment, for example, Windows, Linux, or macOS.
- You'll require an active Azure subscription to do the steps in this walkthrough. If you don't have one, create an Azure subscription by following the steps outlined on the Create your Azure free account today webpage.

Steps

1. Launch the Visual Studio Code editor.
2. The two Visual Studio Code extensions *Azure Account* and *Azure Terraform,* must be installed. To install the first extension, from inside Visual Studio Code, select **File** > **Preferences** > **Extensions**.
3. Search for and install the extension **Azure Account**.



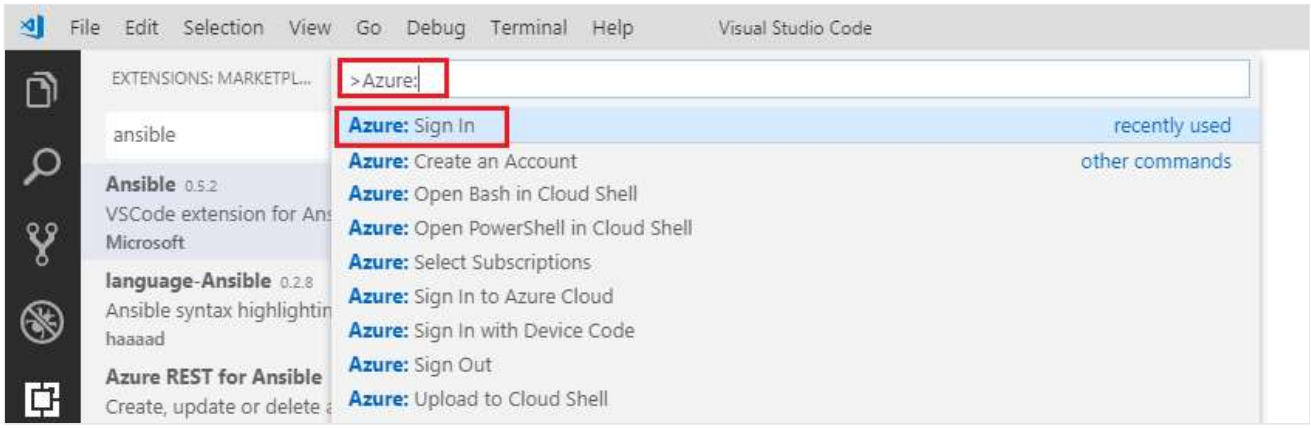
4. Search for and install the extension **Terraform**. Ensure that you select the extension authored by Microsoft, as other authors have similar extensions.



You can view more details of this extension at the Visual Studio Marketplace on the Azure Terraform page.

5. In Visual Studio Code, open the command palette by selecting **View** > **Command Palette**. You can also access the command palette by selecting the **settings** (cog) icon on the bottom, left side of the **Visual Studio Code** window, and then choosing **Command Palette**.

6. In the Command Palette search field, type **Azure:** Select Azure: Sign In from the results.



7. When a browser launches and prompts you to sign in to Azure, select your Azure account. The message *You're signed in now and can close this page*, should display in the browser.

8. Verify that your Azure account now displays at the bottom of the Visual Studio Code window.

9. Create a new file, then copy the following code and paste it into the file.

Yml Copy

```
# Create a resource group if it doesn't exist.
resource "azurerm_resource_group" "myterraformgroup" {
    name     = "terraform-rg2"
    location = "eastus"
```

```
    tags {
        environment = "Terraform Demo"
    }
}

# Create virtual network
resource "azurerm_virtual_network" "myterraformnetwork" {
    name                = "myVnet"
    address_space      = ["10.0.0.0/16"]
    location            = "eastus"
    resource_group_name = "${azurerm_resource_group.myterraformgroup.name}"

    tags {
        environment = "Terraform Demo"
    }
}

# Create subnet
resource "azurerm_subnet" "myterraformsubnet" {
    name                = "mySubnet"
    resource_group_name  = "${azurerm_resource_group.myterraformgroup.name}"
    virtual_network_name = "${azurerm_virtual_network.myterraformnetwork.name}"
    address_prefix      = "10.0.1.0/24"
}

# Create public IPs
resource "azurerm_public_ip" "myterraformpublicip" {
    name                         = "myPublicIP"
    location                     = "eastus"
    resource_group_name          =
"${azurerm_resource_group.myterraformgroup.name}"
    public_ip_address_allocation = "dynamic"

    tags {
        environment = "Terraform Demo"
    }
}

# Create Network Security Group and rule
resource "azurerm_network_security_group" "myterraformnsg" {
    name                = "myNetworkSecurityGroup"
    location            = "eastus"
    resource_group_name = "${azurerm_resource_group.myterraformgroup.name}"

    security_rule {
        name                       = "SSH"
        priority                   = 1001
        direction                   = "Inbound"
        access                     = "Allow"
        protocol                   = "Tcp"
```

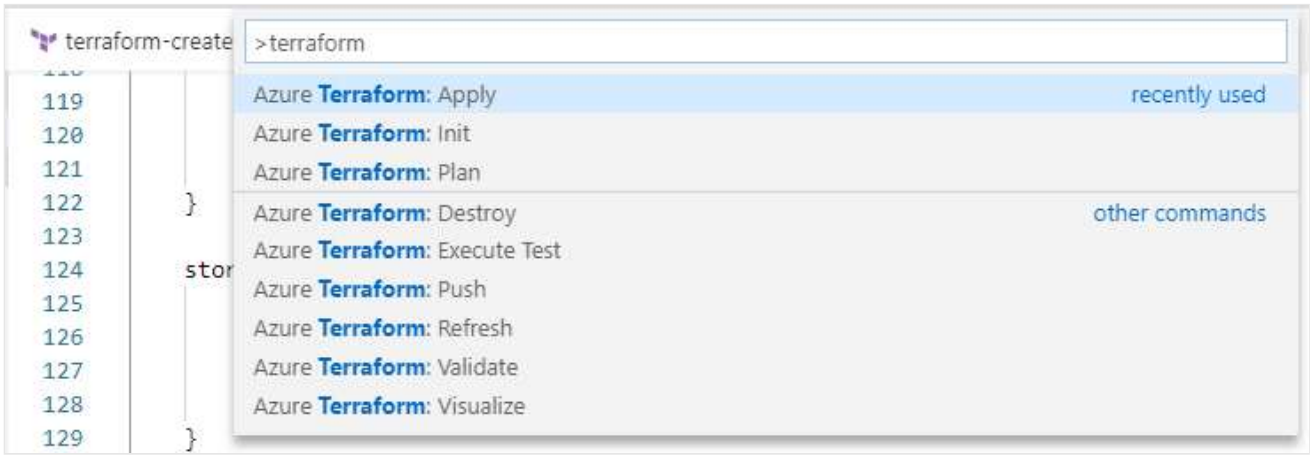
```
        source_port_range          = "*"
        destination_port_range     = "22"
        source_address_prefix      = "*"
        destination_address_prefix = "*"
    }

    tags {
        environment = "Terraform Demo"
    }
}

# Create network interface
resource "azurerm_network_interface" "myterraformnic" {
    name                      = "myNIC"
    location                  = "eastus"
    resource_group_name       = "${azurerm_resource_group.myterraformgroup.name}"
    network_security_group_id =
"${azurerm_network_security_group.myterraformnsg.id}"

    ip_configuration {
        name                          = "myNicConfiguration"
        subnet_id                     = "${azurerm_subnet.myterraformsubnet.id}"
        private_ip_address_allocation = "dynamic"
        public_ip_address_id          =
"${azurerm_public_ip.myterraformpublicip.id}"
    }

    tags {
        environment = "Terraform Demo"
    }
}

# Generate random text for a unique storage account name
resource "random_id" "randomId" {
    keepers = {
        # Generate a new ID only when a new resource group is defined
        resource_group = "${azurerm_resource_group.myterraformgroup.name}"
    }

    byte_length = 8
}

# Create storage account for boot diagnostics
resource "azurerm_storage_account" "mystorageaccount" {
    name                        = "diag${random_id.randomId.hex}"
    resource_group_name         =
"${azurerm_resource_group.myterraformgroup.name}"
    location                    = "eastus"
    account_tier                = "Standard"
    account_replication_type    = "LRS"
```

```
    tags {
        environment = "Terraform Demo"
    }
}

# Create virtual machine
resource "azurerm_virtual_machine" "myterraformvm" {
    name                  = "myVM"
    location              = "eastus"
    resource_group_name   = "${azurerm_resource_group.myterraformgroup.name}"
    network_interface_ids = ["${azurerm_network_interface.myterraformnic.id}"]
    vm_size               = "Standard_DS1_v2"

    storage_os_disk {
        name              = "myOsDisk"
        caching           = "ReadWrite"
        create_option     = "FromImage"
        managed_disk_type = "Premium_LRS"
    }

    storage_image_reference {
        publisher = "Canonical"
        offer     = "UbuntuServer"
        sku       = "16.04.0-LTS"
        version   = "latest"
    }

    os_profile {
        computer_name  = "myvm"
        admin_username = "azureuser"
        admin_password = "Password0134!"
    }

    os_profile_linux_config {
        disable_password_authentication = false
    }

    boot_diagnostics {
        enabled = "true"
        storage_uri =
"${azurerm_storage_account.mystorageaccount.primary_blob_endpoint}"
    }

    tags {
        environment = "Terraform Demo"
    }
}
```

10. Save the file locally with the file name `terraform-createvm.tf`.

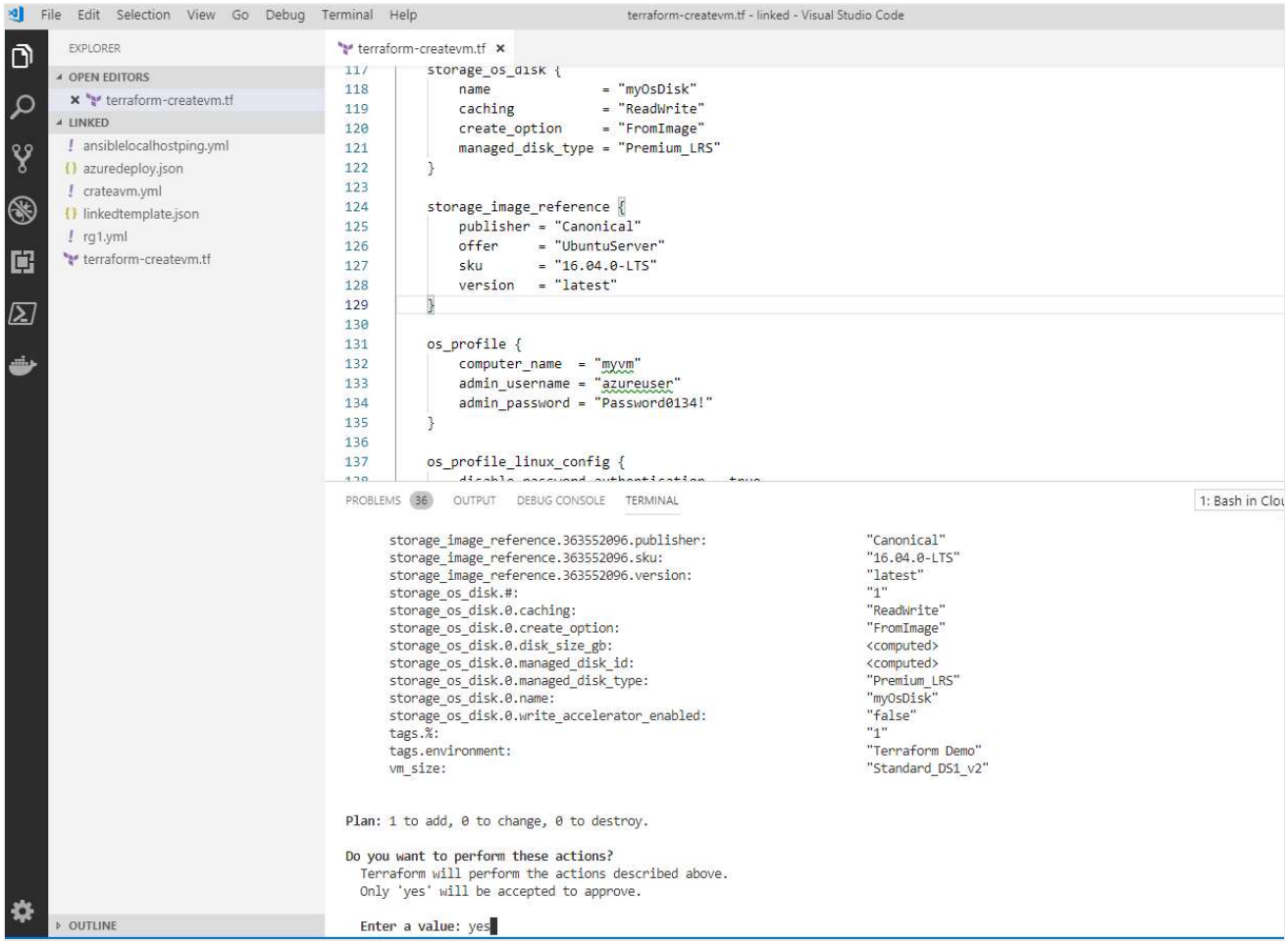
11. In Visual Studio Code, select **View** > **Command Palette**. Search for the command by entering **terraform** into the search field. Select the following command from the dropdown list of commands:

```
Azure Terraform: apply
```



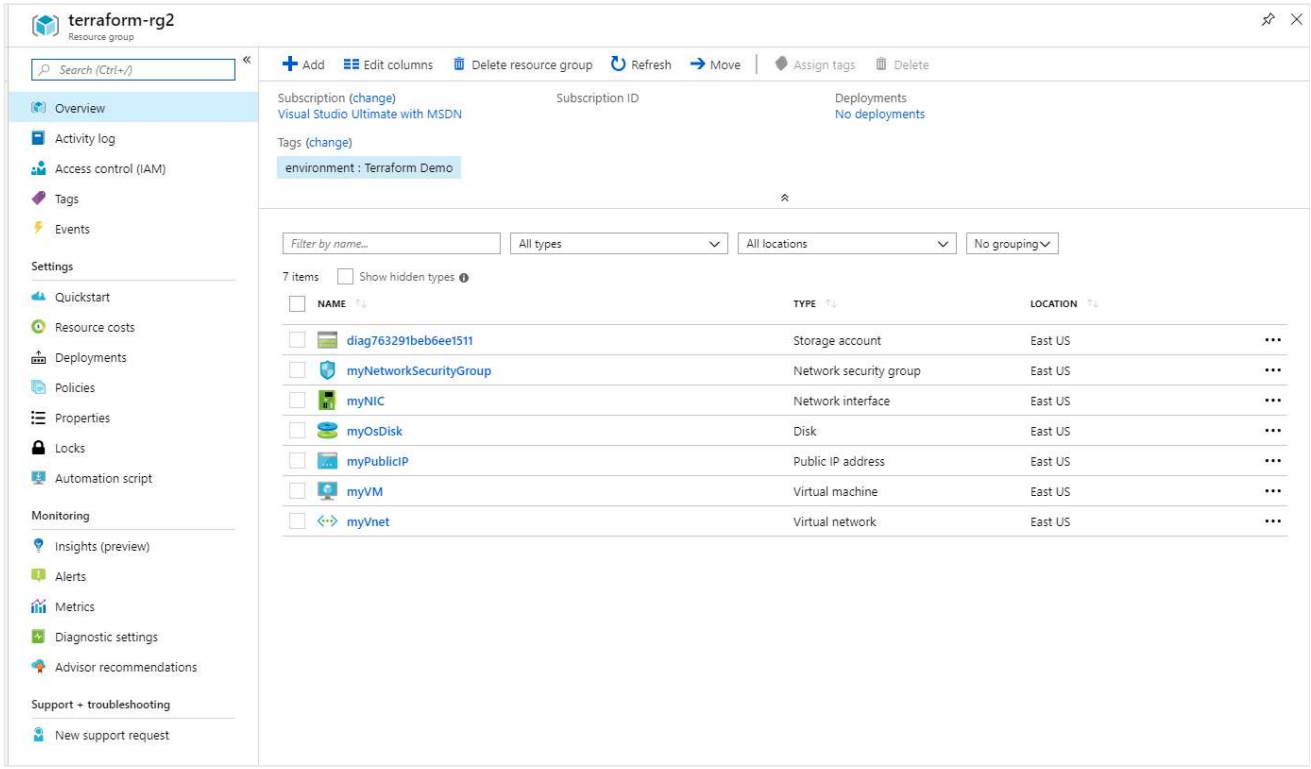
12. If Azure Cloud Shell isn't open in Visual Studio Code, a message might appear in the bottom-left corner asking you if you want to open Azure Cloud Shell. Choose **Accept** and select **Yes**.

13. Wait for the Azure Cloud Shell pane to appear at the bottom of the Visual Studio Code window and start running the file `terraform-createvm.tf`. When you're prompted to apply the plan or cancel, type **Yes**, and then press **Enter**.



14. After the command completes successfully, review the list of resources created.

15. Open the Azure portal and verify the resource group, resources, and the VM has been created. If you have time, sign in with the username and password specified in the .tf config file to verify.



Note

If you wanted to use a public or private key pair to connect to the Linux VM instead of a username and password, you could use the **os_profile_linux_config** module, set the **disable_password_authentication** key value to **true** and include the ssh key details, as in the following code.

yml Copy

```
os_profile_linux_config {
    disable_password_authentication = true
    ssh_keys {
        path     = "/home/azureuser/.ssh/authorized_keys"
        key_data = "ssh-rsa AAAAB3Nz{snip}hwhqT9h"
    }
}
```

You'd also need to remove the password value in the **os_profile module** that present in the example above.

Note

You could also embed the Azure authentication within the script. In that case, you would not need to install the Azure account extension, as in the following example:

yml Copy

```
provider "azurerm" {
    subscription_id = "xxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
    client_id       = "xxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
    client_secret   = "xxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
    tenant_id       = "xxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
    features {}
}
```

Next unit: Knowledge check

Continue >

How are we doing? ☆ ☆ ☆ ☆ ☆