Azure CLI, Resource Manager, Docker

Lecture 04

Deep Azure @ McKesson

Zoran B. Djordjević

Azure CLI

CLI

- Clicking and clacking through Azure Portal is very efficient when you are dealing with one or two services. When dealing with a large number of VMs in a cluster or a complex application requiring many VMs and many services, you need a scripting tool. Azure CLI serves the purpose.
- The Azure CLI 2.0 is Azure's new command line experience for managing Azure resources. We can use it in our browser with Azure Cloud Shell, or you can install Azure CLI 2.0 on MacOS, Linux, and Windows and run it from the command line.
- Azure CLI 2.0 is optimized for managing and administering Azure resources from the command line, and for building automation scripts that work against the Azure Resource Manager.
- Installation instructions are at:

`https://docs.microsoft.com/en-us/cli/azure/install-azure-cli?view=azure-cli-latest#install-on-windows`

Installing Azure CLI

- To install the CLI on Windows and use it in the Windows command-line, download and run the Azure CLI Installer (MSI): azure-cli-2.0.20.msi. Installation is painless.
- On Debian Linux (Ubuntu) do the following:

```
echo "deb [arch=amd64] https://packages.microsoft.com/repos/azure-cli/ wheezy main" |\
sudo tee /etc/apt/sources.list.d/azure-cli.list

sudo apt-key adv --keyserver packages.microsoft.com --recv-keys
52E16F86FEE04B979B07E28DB02C46DF417A0893    # on one line
sudo apt-get install apt-transport-https
sudo apt-get update && sudo apt-get install azure-cli
```

- On RedHat, CentOS, Fedora:
- Import the Microsoft repository key:

```
sudo rpm --import https://packages.microsoft.com/keys/microsoft.asc
```

- Create local azure-cli repository `/etc/yum.repos.d/azure-cli.repo` with information:

```
[azure-cli]
name=Azure CLI
baseurl=https://packages.microsoft.com/yumrepos/azure-cli\nenabled=1
gpgcheck=1
gpgkey=https://packages.microsoft.com/keys/microsoft.asc
```

- Update the yum package index and install:

```
yum check-update
sudo yum install azure-cli
```

- Run the CLI from the command prompt with the az command.

Authenticate our AZ CLI access

- After the installation verify you have `az` command

```
C:\Users\zdjor> where az
C:\Program Files (x86)\Microsoft SDKs\Azure\CLI2\wbin\az.cmd
```

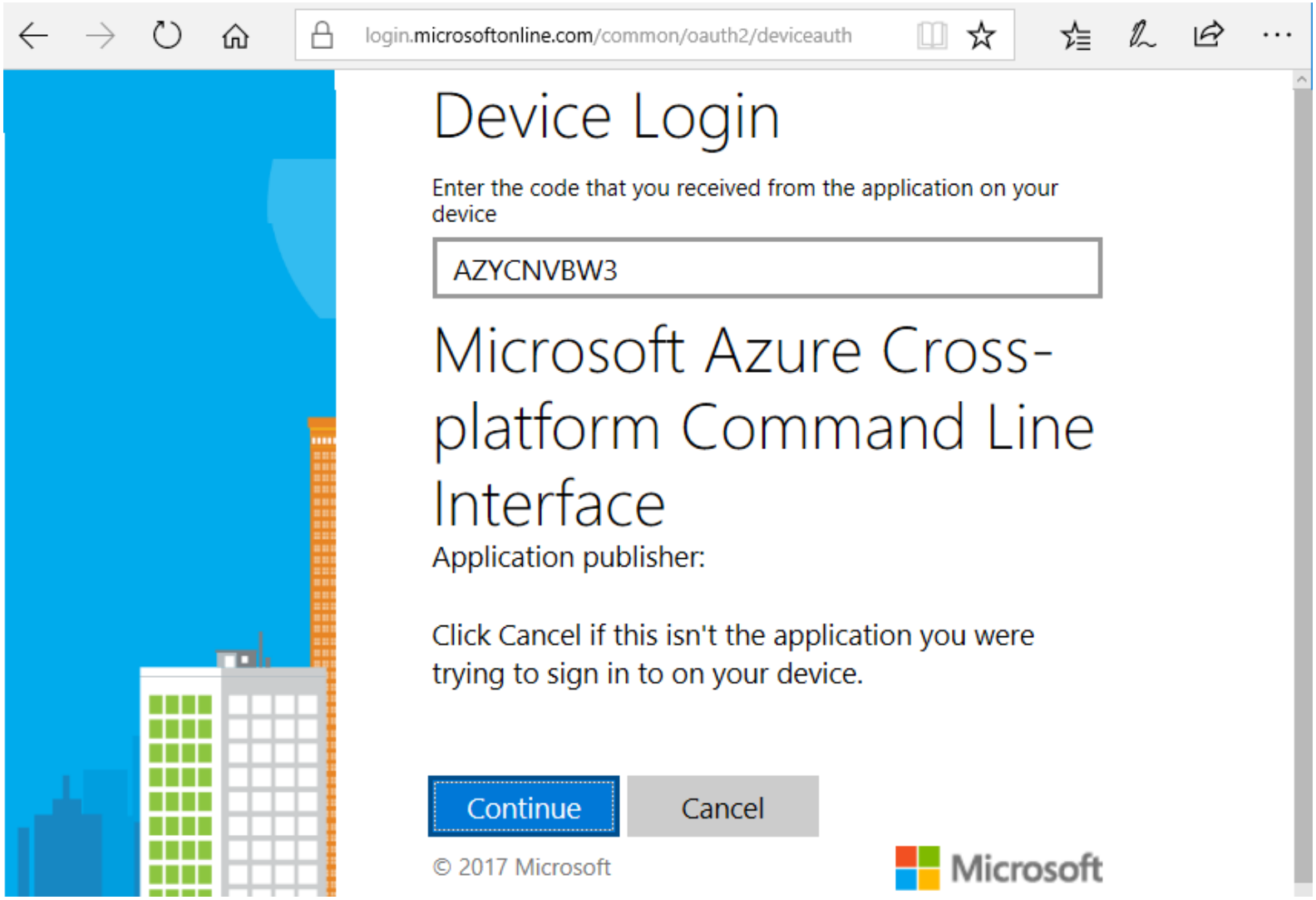
- Try az command with any option

```
C:\Users\zdjor> az loginaz provider list --query "[].{Provider:namespace,
Status:registrationState}" --out table
Please run 'az login' to setup account.
```

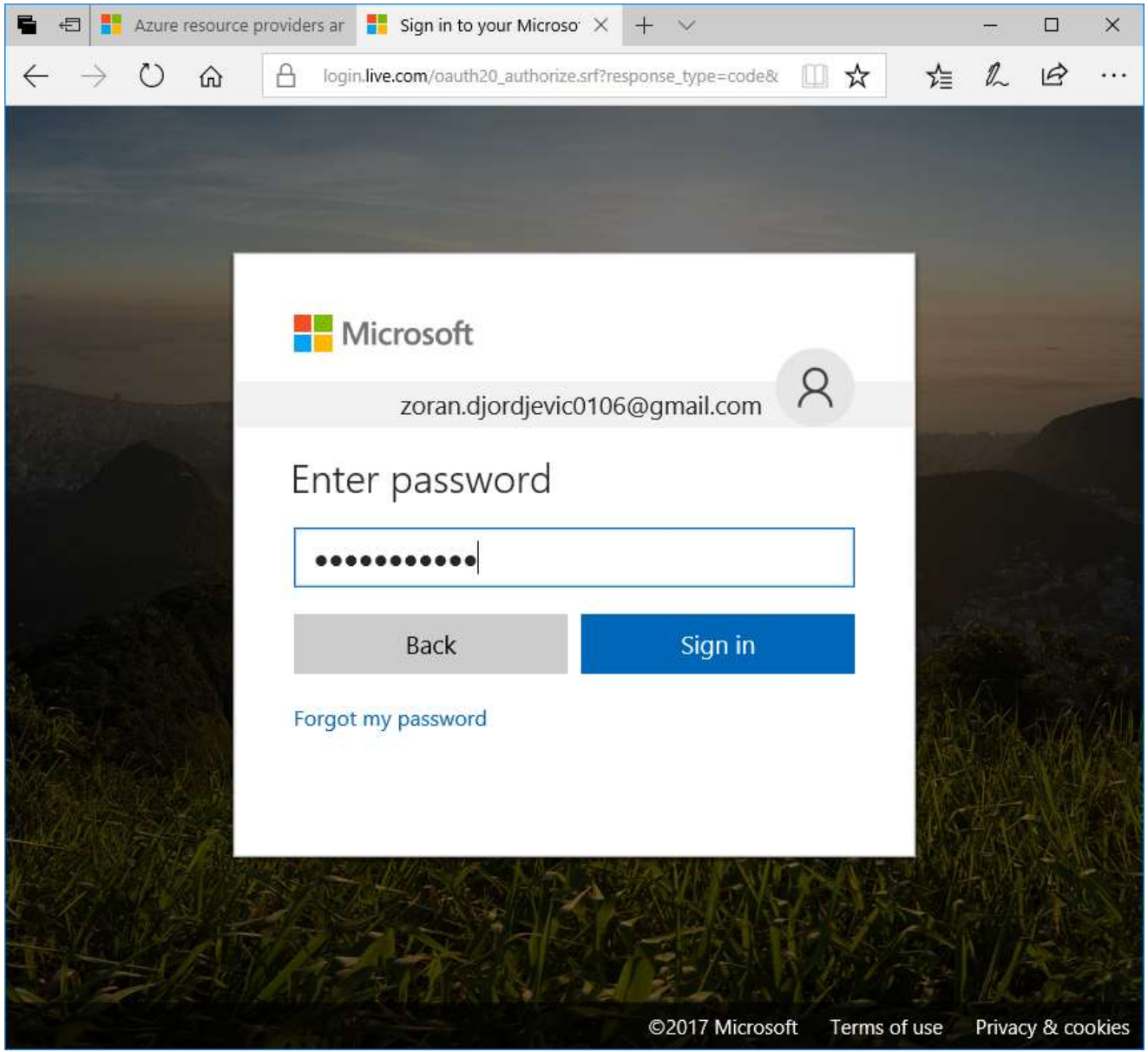
- Do as you are told

```
C:\Users\zdjor> az login
To sign in, use a web browser to open the page https://aka.ms/devicelogin and
enter the code AZYCNVBW3 to authenticate.
```

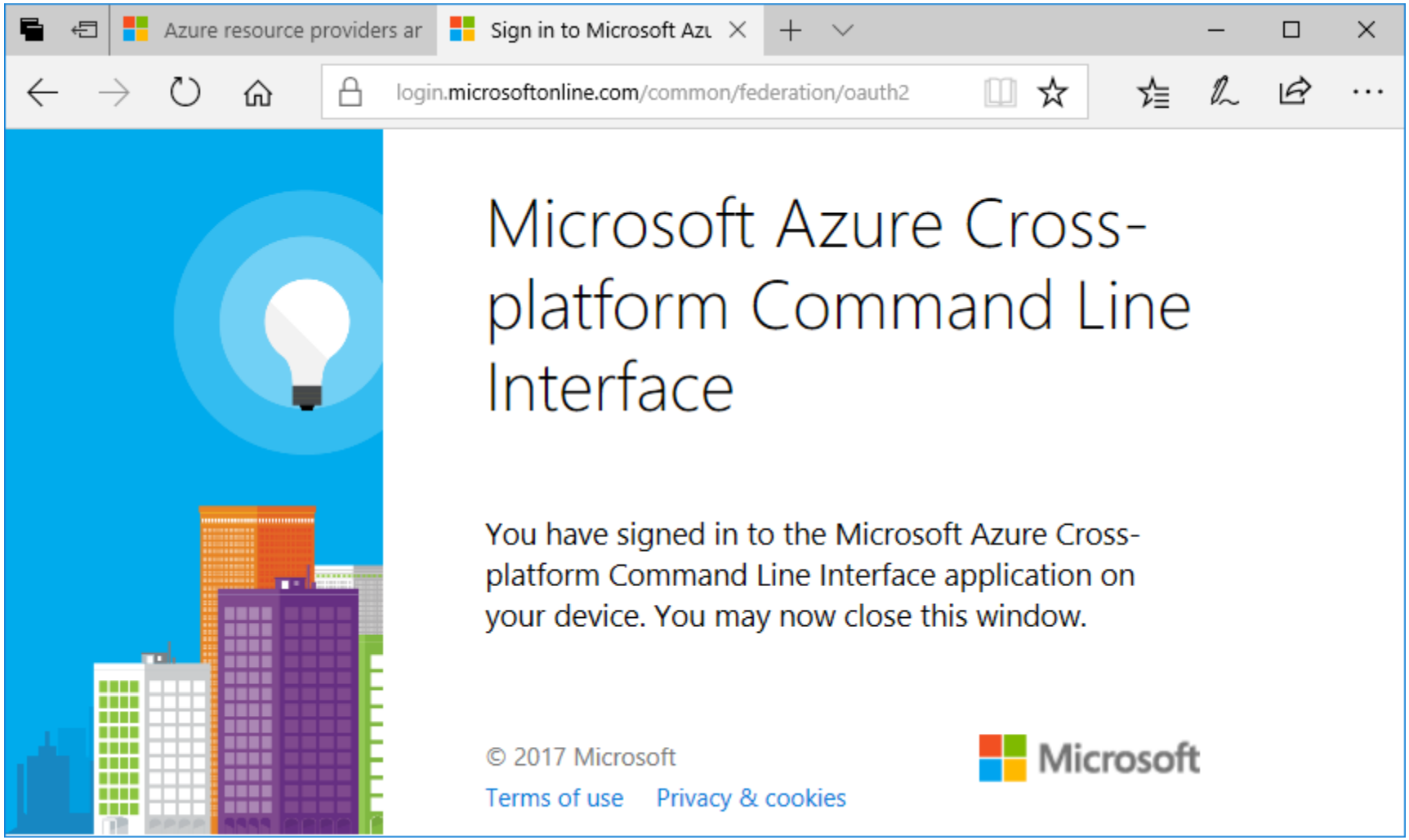
Device Login



Provide the registration email and password



Process is finished



`az login` returns confirmation

```
C:\Users\zdjor>az login
To sign in, use a web browser to open the page
https://aka.ms/devicelogin and enter the code AZYCNVBW3 to
authenticate.
[
  {
    "cloudName": "AzureCloud",
    "id": "ab293589-2b01-4b26-97a4-24f36b894fd2",
    "isDefault": true,
    "name": "Free Trial",
    "state": "Enabled",
    "tenantId": "46519417-c1a8-45a2-8fe9-556f3c5cfa38",
    "user": {
      "name": "zoran.djordjevic0106@gmail.com",
      "type": "user"
    }
  }
]

C:\Users\zdjor>
```

List of providers registered with Resource Manager

```
C:\Users\zdjor>az provider list --query "[].{Provider:namespace,
Status:registrationState}" --out table
Provider                                  Status
----------------------------------------  -------------
Microsoft.Compute                         Registered
Microsoft.ContainerRegistry               Registered
microsoft.insights                        Registered
Microsoft.Network                         Registered
Microsoft.Sql                             Registered
Microsoft.Storage                         Registered
Microsoft.Web                             Registered
84codes.CloudAMQP                         NotRegistered
AppDynamics.APM                           NotRegistered
Aspera.Transfers                          NotRegistered
Auth0.Cloud                               NotRegistered
Citrix.Cloud                              NotRegistered
Cloudyn.Analytics                         NotRegistered
Conexlink.MyCloudIT                       NotRegistered
Crypteron.DataSecurity                    NotRegistered
Dynatrace.DynatraceSaaS                   NotRegistered
Dynatrace.Ruxit                           NotRegistered
LiveArena.Broadcast                       NotRegistered
```

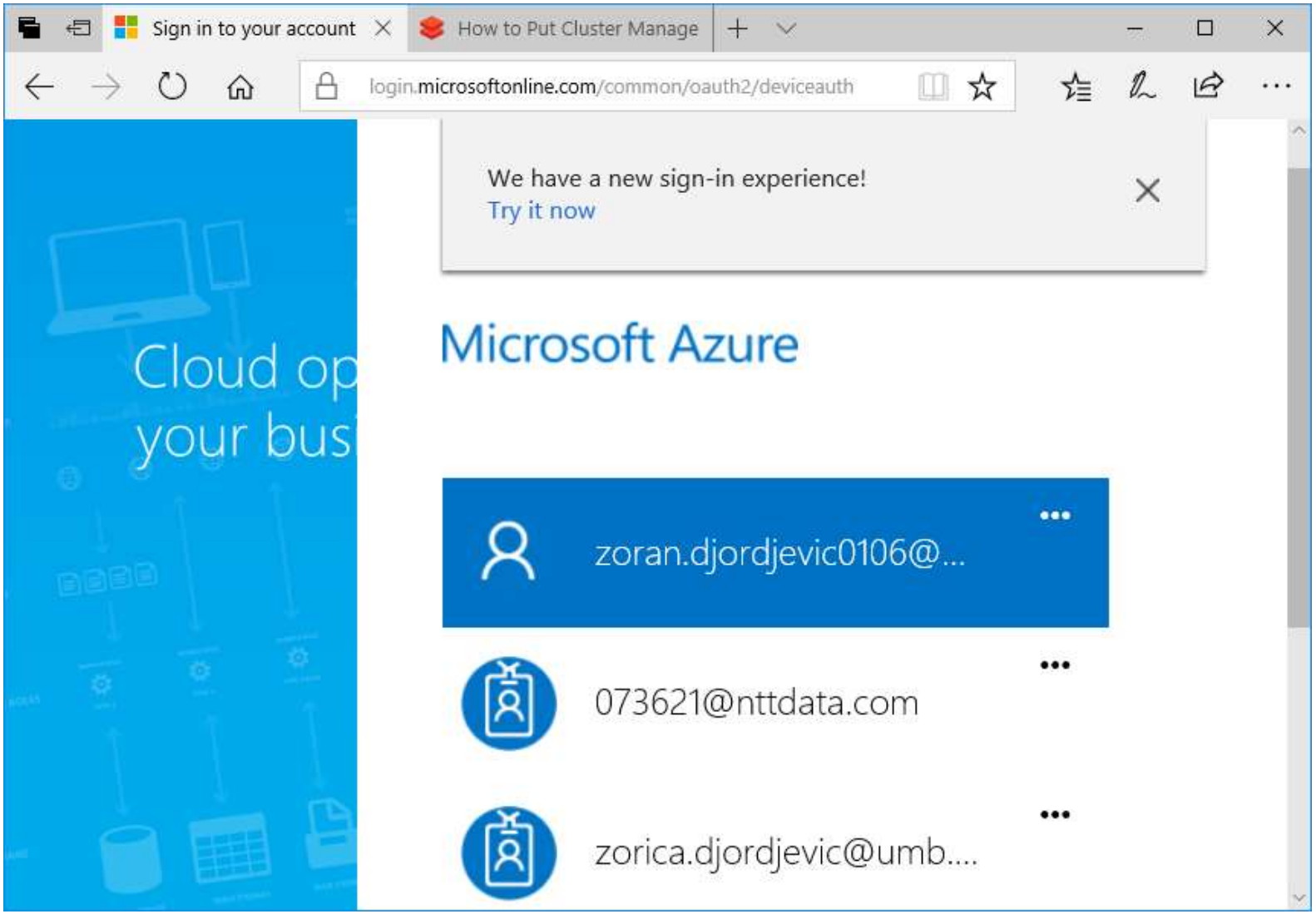
On a Linux box, process is the same

Terminal 5:44 AM

buntu@ubuntu: ~

```
buntu@ubuntu:~$ Users\zdjor>az provider list --query "[].{Provider:namespace, St
atus:registrationState}" --out table
Userszdjor: command not found
buntu@ubuntu:~$ az provider list --query "[].{Provider:namespace, Status:registr
ationState}" --out table
Please run 'az login' to setup account.
buntu@ubuntu:~$ az login
To sign in, use a web browser to open the page https://aka.ms/devicelogin and en
ter the code EPZESCWY7 to authenticate.
```

Provide Azure username, i.e. email



`az login` returns the same JSON as on Windows

```
Terminal                                                    5:47 AM
buntu@ubuntu: ~
buntu@ubuntu:~$ az provider list --query "[].{Provider:namespace, Status:registr
ationState}" --out table
Please run 'az login' to setup account.
buntu@ubuntu:~$ az login
To sign in, use a web browser to open the page https://aka.ms/devicelogin and en
ter the code EPZESCWY7 to authenticate.
^Cbuntu@ubuntu:~$ az login
To sign in, use a web browser to open the page https://aka.ms/devicelogin and en
ter the code EYVFYCZZV to authenticate.
[
  {
    "cloudName": "AzureCloud",
    "id": "ab293589-2b01-4b26-97a4-24f36b894fd2",
    "isDefault": true,
    "name": "Free Trial",
    "state": "Enabled",
    "tenantId": "46519417-c1a8-45a2-8fe9-556f3c5cfa38",
    "user": {
      "name": "zoran.djordjevic0106@gmail.com",
      "type": "user"
    }
  }
]
buntu@ubuntu:~$
```

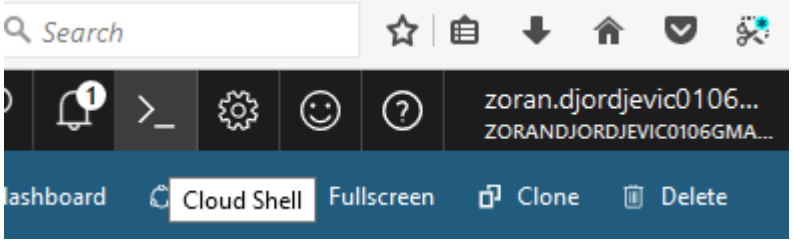
- We are now ready to issue `az` commands.

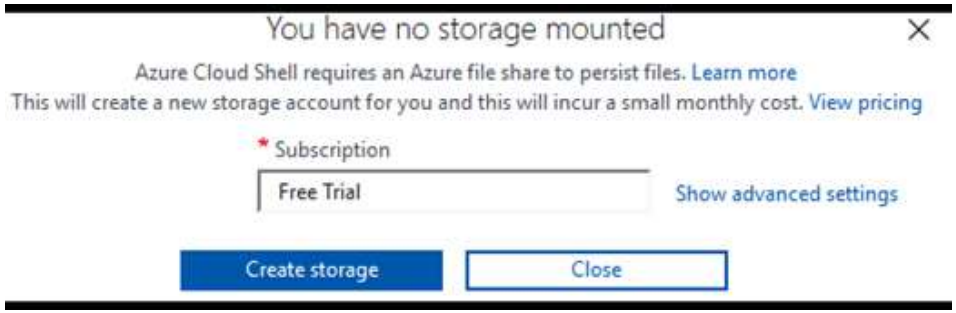
The same output as on Windows

```
Terminal                                                    5:48 AM
buntu@ubuntu: ~
microsoft.aadiam                          NotRegistered
Microsoft.Addons                          NotRegistered
Microsoft.ADHybridHealthService           Registered
Microsoft.Advisor                         NotRegistered
Microsoft.AnalysisServices                NotRegistered
Microsoft.ApiManagement                   NotRegistered
Microsoft.Authorization                   Registered
Microsoft.Automation                      NotRegistered
Microsoft.AzureActiveDirectory            NotRegistered
Microsoft.AzureStack                      NotRegistered
Microsoft.Batch                           NotRegistered
Microsoft.BatchAI                         NotRegistered
Microsoft.Billing                         Registered
Microsoft.BingMaps                        NotRegistered
Microsoft.BizTalkServices                 NotRegistered
Microsoft.Cache                           NotRegistered
Microsoft.Cdn                             NotRegistered
Microsoft.CertificateRegistration         NotRegistered
Microsoft.ClassicCompute                  NotRegistered
Microsoft.ClassicNetwork                  NotRegistered
Microsoft.ClassicStorage                  NotRegistered
Microsoft.ClassicSubscription             Registered
Microsoft.ClassicInfrastructureMigrate    NotRegistered
Microsoft.CognitiveServices               NotRegistered
```

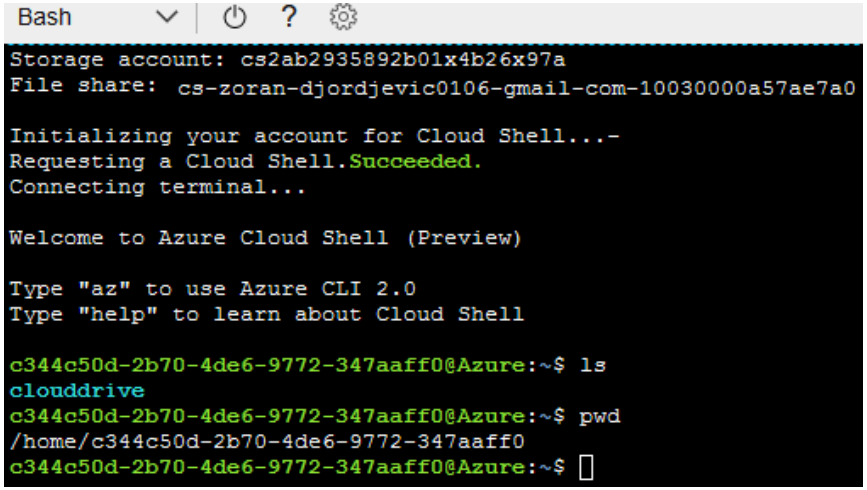
Cloud Shell

- We can run all CLI (Shell) commands in our Portal. Select Shell icon >_





- First time, on the next screen select Bash shell (Linux) or PowerShell (Windows).
- Either way you will be told you have no storage. Create storage!
- After a minute or two you will be presented with a bash shell prompt. It is ours to use.



az --help

- We, of course, read documentation. When in hurry, type:

```
C:\Users\zdjor> az --help
For version info, use 'az --version'
Group
    az
Subgroups:
    account          : Manage Azure subscription information.
    acr              : Manage Azure Container Registries.
    acs              : Manage Azure Container Services.
    ad               : Synchronize on-premises directories and manage Azure Act Dir resources.
    aks              : Manage Kubernetes clusters.
    appservice       : Manage App Service plans.
    backup           : Commands to manage Azure Backups.
    batch            : Manage Azure Batch.
    batchai          : Batch AI.
    billing          : Manage Azure Billing.
    cdn              : Manage Azure Content Delivery Networks (CDNs).
    cloud            : Manage registered Azure clouds.
    cognitiveservices: Manage Azure Cognitive Services accounts.
    component        : Manage and update Azure CLI 2.0 components.
    consumption      : Manage consumption of Azure resources.
    container        : (PREVIEW) Manage Azure Container Instances.
    cosmosdb         : Manage Azure Cosmos DB database accounts.
    disk             : Manage Azure Managed Disks.
    dla              : (PREVIEW) Manage Data Lake Analytics accounts, jobs, and catalogs.
    dls              : (PREVIEW) Manage Data Lake Store accounts and filesystems.
    eventgrid        : Manage Azure Event Grid topics and subscriptions.
    extension        : Manage and update CLI extensions.
    feature          : Manage resource provider features.
    functionapp      : Manage function apps.
    group            : Manage resource groups and template deployments.
    image            : Manage custom virtual machine images.
    iot              : (PREVIEW) Manage Internet of Things (IoT) assets.
    keyvault         : Safeguard and maintain control of keys, secrets, and certificates.
    lab              : Manage Azure DevTest Labs.
    lock             : Manage Azure locks.
```

az --help

```
    lock                : Manage Azure locks.
    managedapp          : Manage template solutions provided & maintained by Independent Software
                          Vendors (ISVs).
    monitor             : Manage the Azure Monitor Service.
    mysql               : Manage Azure Database for MySQL servers.
    network             : Manage Azure Network resources.
    policy              : Manage resource policies.
    postgres            : Manage Azure Database for PostgreSQL servers.
    provider            : Manage resource providers.
    redis               : Access to a secure, dedicated Redis cache for our Azure applications.
    resource            : Manage Azure resources.
    role                : Manage user roles for access control with Azure AD and service principals.
    sf                  : Manage and administer Azure Service Fabric clusters.
    snapshot            : Manage point-in-time copies of managed disks, native blobs, or other
                          snapshots.
    sql                 : Manage Azure SQL Databases and Data Warehouses.
    storage             : Manage Azure Cloud Storage resources.
    tag                 : Manage resource tags.
    vm                  : Provision Linux or Windows virtual machines.
    vmss                : Manage groupings of virtual machines in an Virtual Machine Scale Set(VMSS).
    webapp              : Manage web apps.
Commands:
    configure           : Display and manage the Azure CLI 2.0 configuration. This command is
                          interactive.
    feedback            : Loving or hating the CLI?  Let us know!
    find                : Find Azure CLI commands.
    interactive         : Start interactive mode.
    login               : Log in to Azure.
    logout              : Log out to remove access to Azure subscriptions.
```

az account --help

- Every subgroup has it own options. For example:

```
C:\Users\zdjor> az account --help
Group
    az account: Manage Azure subscription information.
Subgroups:
    lock            : Manage Azure subscription level locks.
Commands:
    clear           : Clear all subscriptions from the CLI's local cache.
    get-access-token: Get a token for utlilities to access Azure.
    list            : Get a list of subscriptions for the logged in account.
    list-locations  : List supported regions for the current subscription.
    set             : Set a subscription to be the current active subscription.
    show            : Get the details of a subscription.

C:\Users\zdjor> az group --help
Group
    az group: Manage resource groups and template deployments.
Subgroups:
    deployment: Manage Azure Resource Manager deployments.
    lock      : Manage Azure resource group locks.
Commands:
    create    : Create a new resource group.
    delete    : Delete a resource group.
    exists    : Check if a resource group exists.
    export    : Captures a resource group as a template.
    list      : List resource groups.
    show      : Gets a resource group.
    update    : Update a resource group.
    wait      : Place the CLI in waiting state until condition of resource group is met.
```

Find proper names of Azure Regions

```
C:\Users\zdjor> az account list-locations --out table
DisplayName            Latitude    Longitude  Name
-------------------  ----------  -----------  ------------------
East Asia               22.267      114.188   eastasia
Southeast Asia           1.283      103.833   southeastasia
Central US              41.5908     -93.6208  centralus
East US                 37.3719     -79.8164  eastus
East US 2               36.6681     -78.3889  eastus2
West US                 37.783     -122.417   westus
North Central US        41.8819     -87.6278  northcentralus
South Central US        29.4167     -98.5     southcentralus
North Europe            53.3478      -6.2597  northeurope
West Europe             52.3667       4.9     westeurope
Japan West              34.6939     135.502   japanwest
Japan East              35.68       139.77    japaneast
Brazil South           -23.55       -46.633   brazilsouth
Australia East         -33.86       151.209   australiaeast
Australia Southeast    -37.8136     144.963   australiasoutheast
South India             12.9822      80.1636  southindia
Central India           18.5822      73.9197  centralindia
West India              19.088       72.868   westindia
Canada Central          43.653      -79.383   canadacentral
Canada East             46.817      -71.217   canadaeast
UK South                50.941       -0.799   uksouth
UK West                 53.427       -3.084   ukwest
West Central US         40.89      -110.234   westcentralus
West US 2               47.233     -119.852   westus2
Korea Central           37.5665     126.978   koreacentral
Korea South             35.1796     129.076   koreasouth
```

Create a Resource Group

- First, create a Resource Group. Resource Groups in Azure provide a way to manage multiple resources. We have seen how a resource group organized all components of our Web Application.
- Create a resource group named "MyResourceGroup" in the *southcentral* region of Azure. To do so type the following command:

```
C:\>  az group create -n MyResourceGroup -l southcentralus
{
  "id": "/subscriptions/ab293589-2b01-4b26-97a4-24f36b894fd2/resourceGroups/MyResourceGroup",
  "location": "southcentralus",
  "managedBy": null,
  "name": "MyResourceGroup",
  "properties": {
    "provisioningState": "Succeeded"
  },
  "tags": null
}
```

Create a Linux VM

- Now that we have our resource group, we create a Linux VM within it.
- We will create a Linux VM using UbuntuLTS image, with two attached storage disks of 10 GB and 20 GB, with the following command:

```
C:..\> az vm create -n MyLinuxVM -g MyResourceGroup --image
UbuntuLTS --data-disk-sizes-gb 10 20
```

- When you run the preceding command, the Azure CLI 2.0 looks for an SSH key pair stored under our `~/.ssh` directory. We did create that key under Cygwin but perhaps not under Windows.
- Rerun the above command with `--generate-ssh-keys`

```
C:\..> az vm create -n MyLinuxVM -g MyResourceGroup --image
UbuntuLTS --data-disk-sizes-gb 10 20 --generate-ssh-keys
```

```
SSH key files 'C:\Users\zdjor\.ssh\id_rsa' and
'C:\Users\zdjor\.ssh\id_rsa.pub' have been generated under ~/.ssh
to allow SSH access to the VM. If using machines without
permanent storage, back up our keys to a safe location.
```

- `| Running ...`
- If you examine our Windows (Linux) home directory you will see `.ssh` directory with 2 files: id_rsa and `id_rsa.pub` both have Unix permissions `-rwx------`

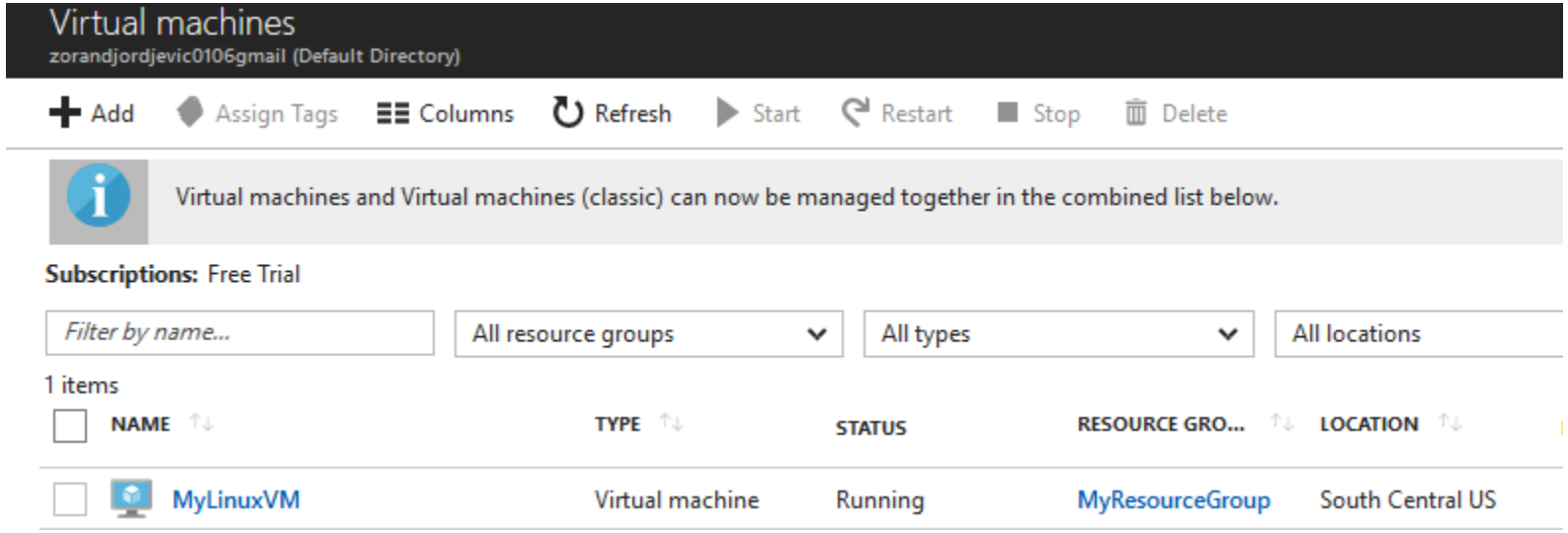
VM is there, eventually

- After several minutes, you will get an output that looks like this:

```
{/ Finished ..
  "fqdns": "",
  "id": "/subscriptions/ab293589-2b01-4b26-97a4-24f36b894fd2/resourceGroups/MyResourceGroup/providers/Microsoft.Compute/virtualMachines/MyLinuxVM",
  "location": "southcentralus",
  "macAddress": "00-0D-3A-75-2F-67",
  "powerState": "VM running",
  "privateIpAddress": "10.0.0.4",
  "publicIpAddress": "13.65.81.33",
  "resourceGroup": "MyResourceGroup",
  "zones": ""
}
```

This is the public IP address. We could connect to

- Up in Azure Portal, we would see:



Use `ssh` to connect to our Linux VM

- This new VM was created with new RSA keys. Open our Cygwin, navigate to the .ssh directory in the home directory of our Windows user (`C:\Users\zdjor\.ssh` , in my case) and copy newly created keys to our Cygwin's home directory

```
$ cp id_rsa* ~/.ssh
zdjor@DESKTOP-0NUU7AF /cygdrive/c/Users/zdjor/.ssh
```

- Now, we can ssh into new machine. Notice that we are using the name of the local users

```
$ ssh 13.65.81.33
The authenticity of host '13.65.81.33 (13.65.81.33)' can't be established.
ECDSA key fingerprint is SHA256:QrFfFggAcHPdjW6HbF6LVxcZIXvT/wqJw8o1TsNdfEw.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '13.65.81.33' (ECDSA) to the list of known hosts.
Welcome to Ubuntu 16.04.3 LTS (GNU/Linux 4.11.0-1013-azure x86_64)
 * Documentation:  https://help.ubuntu.com
 . . . .
 * Support:        https://ubuntu.com/advantage
  Get cloud support with Ubuntu Advantage Cloud Guest:
    http://www.ubuntu.com/business/services/cloud
. . . . .
Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

zdjor@MyLinuxVM:~$ pwd
/home/zdjor
```

az vm create --help

```
C:\Users\zdjor> az vm create --help
Command
    az vm create: Create an Azure Virtual Machine.
        For an end-to-end tutorial, see https://docs.microsoft.com/azure/virtual-machines/virtual-
        machines-linux-quick-create-cli.
Arguments
    --name -n                [Required]: Name of the virtual machine.
    --resource-group -g      [Required]: Name of resource group. We can configure the default group
                                         using `az configure --defaults group=<name>`.
    --attach-data-disks                : Attach existing data disks to the VM. Can use the name or ID of a managed
                                         disk or the URI to an unmanaged disk VHD.
    --attach-os-disk                   : Attach an existing OS disk to the VM. Can use the name or ID of a managed
                                         disk or the URI to an unmanaged disk VHD.
    --availability-set                 : Name or ID of an existing availability set to add the VM to. Default None
    --custom-data                      : Custom init script file or text (cloud-init, cloud-config, etc..).
    --image                            : The name of the operating system image as a URN alias, URN,
                                         custom image name or ID, or VHD blob URI. This parameter is
                                         required unless using `--attach-os-disk.`.Values from: az vm image list, az
                                         vm image show.
    --license-type                     : License type if the Windows image or disk used was licensed  on-premises.
                                         Allowed values: Windows_Client, Windows_Server.
    --location -l                      : Location in which to create VM and related resources. If  default location
                                         is not configured, will default to the
                                         resource group's location.
    --no-wait                          : Do not wait for the long running operation to finish.
    --secrets                          : One or many Key Vault secrets as JSON strings or files via
                                         `@<file path>` containing `[{ "sourceVault": { "id": "value" }, "vaultCertificates": [{
"certificateUrl":
                                         "value", "certificateStore": "cert store name (only on windows)"}] }]`.
    --size                             : The VM size to be created. See
                                         https://azure.microsoft.com/en-us/pricing/details/virtual-machines/ for size info.
Default: Standard_DS1_v2.
    --tags                             : Space separated tags in 'key[=value]' format. Use "" to
                                         clear existing tags.
    --validate                         : Generate and validate the ARM template without creating any  resources.
    --zone -z                          : Availability zone into which to provision the resource.
                                         Allowed values: 1, 2, 3.
```

az vm create --help

```
Authentication Arguments
    --admin-password                  : Password for the VM if authentication type is 'Password'.
    --admin-username                  : Username for the VM.  Default: zdjor.
    --authentication-type             : Type of authentication to use with the VM. Defaults to
                                        password for Windows and SSH public key for Linux.  Allowed
                                        values: password, ssh.
    --generate-ssh-keys               : Generate SSH public and private key files if missing. The
                                        keys will be stored in the ~/.ssh directory.
    --ssh-dest-key-path               : Destination file path on the VM for the SSH key.
    --ssh-key-value                   : The SSH public key or public key file path.
Managed Service Identity Arguments
    --assign-identity                 : Enables the VM/VMSS to autonomously, using its own managed
                                        identity, to directly authenticate and interact with other
                                        Azure services using bearer tokens.
    --role                            : Role name or id the managed identity will be assigned.
                                        Default: Contributor.
    --scope                           : The scope the managed identity has access to.
Network Arguments
    --asgs                            : Space separated list of existing application security
                                        groups to associate with the VM.
    --nics                            : Names or IDs of existing NICs to attach to the VM. The
                                        first NIC will be designated as primary. If omitted, a new
                                        NIC will be created. If an existing NIC is specified, do
                                        not specify subnet, vnet, public IP or NSG.
    --nsg                             : The name to use when creating a new Network Security Group
                                        (default) or referencing an existing one. Can also
                                        reference an existing NSG by ID or specify "" for none.
    --nsg-rule                        : NSG rule to create when creating a new NSG. Defaults to
                                        open ports for allowing RDP on Windows and allowing SSH on
                                        Linux.  Allowed values: RDP, SSH.
    --private-ip-address              : Static private IP address (e.g. 10.0.0.5).
    --public-ip-address               : Name of the public IP address when creating one (default)
                                        or referencing an existing one. Can also reference an
                                        existing public IP by ID or specify "" for None.
    --public-ip-address-allocation    : Allowed values: dynamic, static.  Default: dynamic.
```

az vm create --help

```
--public-ip-address-dns-name           : Globally unique DNS name for a newly created Public IP.
    --subnet                           : The name of the subnet when creating a new VNet or referencing an existing
                                         one. Can also reference an existing subnet by ID. If omitted, an appropriate
                                         VNet and subnet will be created automatically, or a new one will be created.
    --subnet-address-prefix            : The subnet IP address prefix to use when creating a new VNet in CIDR format.
Default: 10.0.0.0/24.
    --vnet-address-prefix              : The IP address prefix to use when creating a new VNet in
                                         CIDR format.  Default: 10.0.0.0/16.
    --vnet-name                        : Name of the virtual network when creating a new one or
                                         referencing an existing one.
Storage Arguments
    --data-disk-caching                : Storage caching type for the VM data disk(s).  Allowed
                                         values: None, ReadOnly, ReadWrite.
    --data-disk-sizes-gb               : Space separated empty managed data disk sizes in GB to
                                         create.
    --os-disk-caching --storage-caching: Storage caching type for the VM OS disk.  Allowed values:
                                         ReadOnly, ReadWrite.
    --os-disk-name                     : The name of the new VM OS disk.
    --os-disk-size-gb                  : The size of the os disk in GB.
    --os-type                          : Type of OS installed on a custom VHD. Do not use when specifying an URN or
                                         URN alias.  Allowed values: linux, windows.
    --storage-account                  : Only applicable when use with '--use-unmanaged-disk'. The name to use when
creating a new storage account or
                                         referencing an existing one. If omitted, an appropriate storage account in
the same resource group and location
                                         will be used, or a new one will be created.
    --storage-container-name           : Only applicable when use with '--use-unmanaged-disk'. Name of the storage
container for the VM OS disk. Default: vhds.
    --storage-sku                      : The sku of storage account to persist VM. By default, only
                                         Standard_LRS and Premium_LRS are allowed. Using with --use- unmanaged-disk,
all are available.  Allowed values:
                                         Premium_LRS, Standard_GRS, Standard_LRS, Standard_RAGRS,  Standard_ZRS.
    --use-unmanaged-disk               : Do not use managed disk to persist VM.
Global Arguments
    --debug                            : Increase logging verbosity to show all debug logs.
    --help -h                          : Show this help message and exit.
    --output -o                        : Output format.  Allowed values: json, jsonc, table, tsv. Default: json.
    --query                            : JMESPath query string. http://jmespath.org/ for information and examples.
    --verbose                          : Increase logging verbosity. Use --debug for full debug logs.
```

az vm create –help, examples

```
Examples
    Create a default Ubuntu VM with automatic SSH authentication.
        az vm create -n MyVm -g MyResourceGroup --image UbuntuLTS

    Create a default Windows Server VM with a private IP address.
        az vm create -n MyVm -g MyResourceGroup --public-ip-address "" --image Win2012R2Datacenter

    Create a VM from a custom managed image.
        az vm create -g MyResourceGroup -n MyVm --image MyImage

    Create a VM by attaching to a managed operating system disk.
        az vm create -g MyResourceGroup -n MyVm --attach-os-disk MyOsDisk --os-type linux

    Create an Ubuntu Linux VM using a cloud-init script for configuration. See:
    https://docs.microsoft.com/azure/virtual-machines/virtual-machines-linux-using-cloud-init.
        az vm create -g MyResourceGroup -n MyVm --image debian --custom_data MyCloudInitScript.yml

    Create a Debian VM with SSH key authentication and a public DNS entry, located on an existing
    virtual network and availability set.
        az vm create -n MyVm -g MyResourceGroup --image debian --vnet-name MyVnet --subnet subnet1 \
            --availability-set MyAvailabilitySet --public-ip-address-dns-name MyUniqueDnsName \
            --ssh-key-value @key-file
```

az vm create –help, examples

```
Create a Debian VM with SSH key authentication and a public DNS entry, located on an existing
virtual network and availability set.
    az vm create -n MyVm -g MyResourceGroup --image debian --vnet-name MyVnet --subnet subnet1 \
        --availability-set MyAvailabilitySet --public-ip-address-dns-name MyUniqueDnsName \
        --ssh-key-value @key-file

Create a simple Ubuntu Linux VM with a public IP address, DNS entry, two data disks (10GB and
20GB), and then generate ssh key pairs.
    az vm create -n MyVm -g MyResourceGroup --public-ip-address-dns-name MyUniqueDnsName \
        --image ubuntults --data-disk-sizes-gb 10 20 --size Standard_DS2_v2 \
        --generate-ssh-keys
Create a Debian VM using Key Vault secrets.
    az keyvault certificate create --vault-name vaultname -n cert1 \
      -p "$(az keyvault certificate get-default-policy)"
    secrets=$(az keyvault secret list-versions --vault-name vaultname \
      -n cert1 --query "[?attributes.enabled].id" -o tsv)

    vm_secrets=$(az vm format-secret -s "$secrets")

    az vm create -g group-name -n vm-name --admin-username deploy  \
      --image debian --secrets "$vm_secrets"
Create a CentOS VM with Managed Service Identity. The VM will have a 'Contributor' role with
access to a storage account.
    az vm create -n MyVm -g MyResourceGroup --image centos --assign-identity --scope
    /subscriptions/99999999-1bf0-4dda-
    aec3-cb9272f09590/MyResourceGroup/myRG/providers/Microsoft.Storage/storageAccounts/storage1
Create a VM in an availability zone in the current resource group's region
    az vm create -n MyVm -g MyResourceGroup --image Centos --zone 1
```

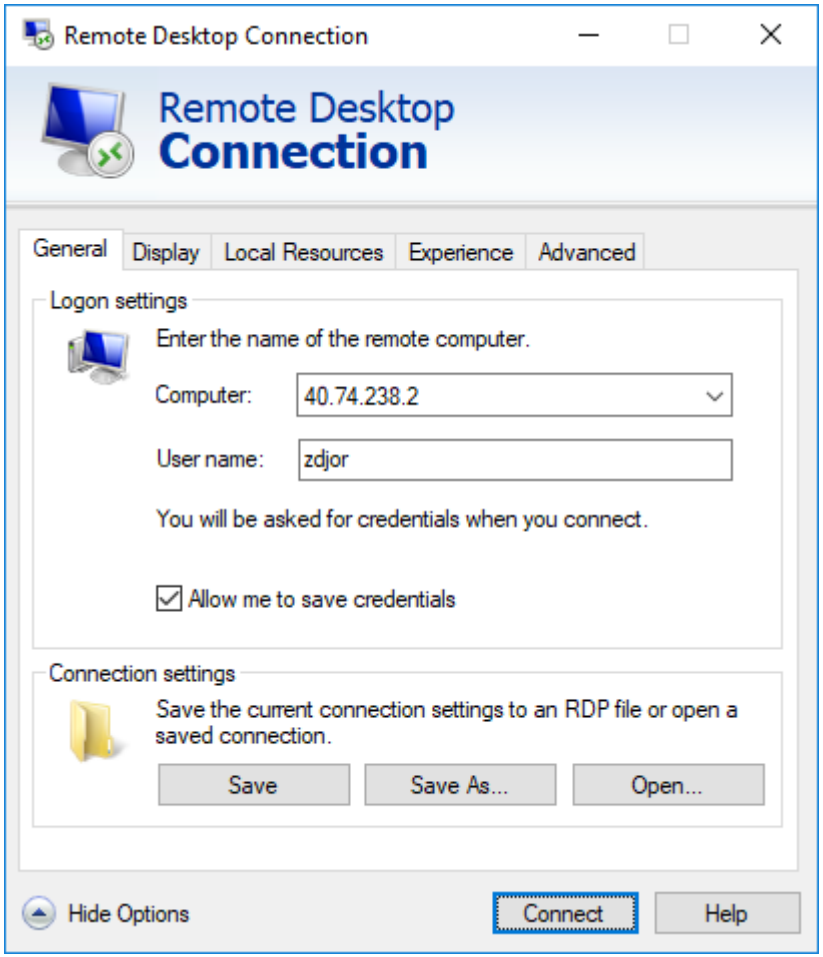
Create Windows VM using CLI

- Create a Windows Server 2016 VM using the `az vm create` command and add it to the same "MyResourceGroup" resource group.
- Like the Linux VM example, we'll also attach two storage disks using the `--data-disk-sizes-gb parameter`.
- Windows servers require usernames/passwords. There are specific rules for what characters can be used as well as the minimum length of both username and password.
- We will be prompted to enter you password when running this command.

```
C:\Users\zdjor>az vm create -n MyWinVM -g MyResourceGroup --image
Win2016Datacenter --data-disk-sizes-gb 10 20
Admin Password:
Confirm Admin Password:
The password length must be between 12 and 123
C:\Users\zdjor>az vm create -n MyWinVM -g MyResourceGroup --image
Win2016Datacenter --data-disk-sizes-gb 10 20
Admin Password:
Confirm Admin Password:
 / Running ...
```

Connect to our Windows VM

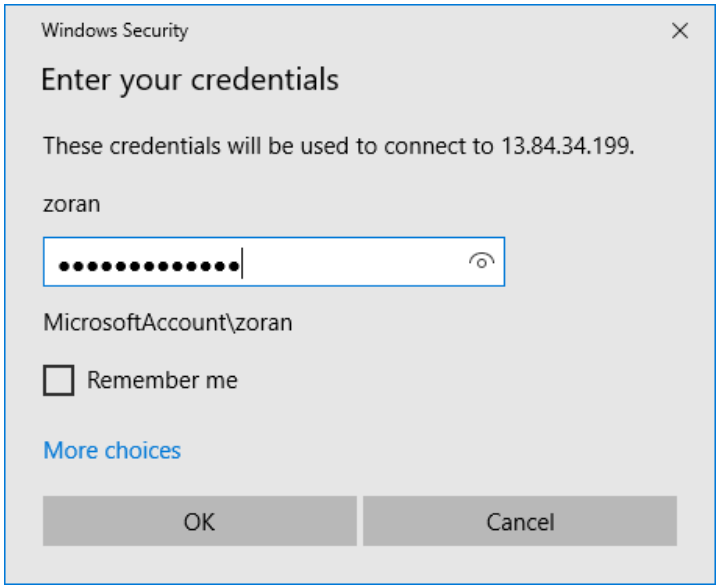

```
{\ Finished ..
  "fqdns": "",
  "id": "/subscriptions/ab293589-2b01-4b26-97a4-24f36b894fd2/resourceGroups/MyResourceGroup/providers/Microsoft.Compute/virtualMachines/MyWinVM",
  "location": "southcentralus",
  "macAddress": "00-0D-3A-71-25-77",
  "powerState": "VM running",
  "privateIpAddress": "10.0.0.5",
  "publicIpAddress": "40.74.238.2",
  "resourceGroup": "MyResourceGroup",
  "zones": ""
}
```

- It would not let me connect.
- I deleted VM and created new one this time by specifying admin-username and admin-password.

Try Again, this time with new credentials

```
C:\Users\zdjor> az vm create -n MyWinVM -g MyResourceGroup --admin-username zoran --admin-password Bost0N123456$ --image Win2016Datacenter --data-disk-sizes-gb 10 20
{/ Finished ..
  "fqdns": "",
  "id": "/subscriptions/ab293589-2b01-4b26-97a4-24f36b894fd2/resourceGroups/MyResourceGroup/providers/Microsoft.Compute/virtualMachines/MyWinVM",
  "location": "southcentralus",
  "macAddress": "00-0D-3A-71-25-77",
  "powerState": "VM running",
  "privateIpAddress": "10.0.0.5",
  "publicIpAddress": "13.84.34.199",
  "resourceGroup": "MyResourceGroup",
  "zones": ""
}
C:\Users\zdjor>where mstsc
C:\Windows\System32\mstsc.exe
C:\Users\zdjor>mstsc /v:13.84.34.199
```



- RDP window that opens lets me enter new admin-username and password and connect.

`--no-wait`

- When creating resources using the Azure CLI 2.0, the az <resource type name> create command waits until the resource has been created and is ready for you to use.
- For example, if you create a VM, the az vm create command will, by default, not return until the VM is created and is ready for you to SSH or RDP into it.
- This behavior makes it easier to write automation scripts that contain multiple steps with dependencies (and prior tasks have to be completed before process continues).
- If you do not need to wait on creation of a resource before doing something else, you can use the `--no-wait` option to start a create action in the background. We can continue using the CLI for other commands.
- For example, the following usage of the az vm create starts a VM deployment and then return much more quickly (and before the VM has fully booted):

```
az vm create -n MyLinuxVM2 -g MyResourceGroup --image UbuntuLTS --no-wait
```

- Using the --no-wait approach helps you optimize our time and sometimes the performance of our automation scripts.

Listing Resources

- The following command shows all VMs we own:

```
C:\Users\zdjor>az vm list --out table
Name        ResourceGroup    Location
---------   ---------------  --------------
MyLinuxVM   MYRESOURCEGROUP  southcentralus
MyWinVM     MYRESOURCEGROUP  southcentralus
```

- Without `--out table` option you would get a long JSON with all the details
- Similarly:

```
C:\Users\zdjor>az group list --out table
Name                         Location        Status
---------------------------  --------------  ---------
cloud-shell-storage-eastus   eastus          Succeeded
MyResourceGroup              southcentralus  Succeeded
```

- The *tsv* output option can be used to get a text-based, tab-separated format without any headers. This format is useful when you want to pipe the output into another text-based tool like grep

```
C:\Users\zdjor>az vm list --output tsv
None    None            /subscriptions/ab293589-2b01-4b26-97a4-
24f36b894fd2/resourceGroups/MYRESOURCEGROUP/providers/Microsoft.Compute/virtualMa
chines/MyLinuxVM        southcentralus  MyLinuxVM               Succeeded
MYRESOURCEGROUP                    Microsoft.Compute/virtualMachines
78957e67-f997-44b3-962e-49f146338540    None
None    None            /subscriptions/ab293589-2b01-4b26-97a4-
24f36b894fd2/resourceGroups/MYRESOURCEGROUP/providers/Microsoft.Compute/virtualMa
chines/MyWinVM  southcentralus  MyWinVM            Succeeded
MYRESOURCEGROUP None                    Microsoft.Compute/virtualMachines
14413ce7-ee58-4262-a228-483eb569342d    None
```

Querying Resources, shaping the output of `list`

- If you type: `az vm list –help`, you will get:

```
C:\Users\zdjor>az vm list --help
Command
    az vm list: List details of Virtual Machines.
        For more information on querying information about Virtual Machines, see
        https://docs.microsoft.com/en-us/cli/azure/query-az-cli2.
Arguments
    --resource-group -g: Name of resource group. We can configure the default group using `az
                         configure --defaults group=<name>`.
    --show-details -d  : Show public ip address, FQDN, and power states. command will run slow.
Global Arguments
    --debug            : Increase logging verbosity to show all debug logs.
    --help -h          : Show this help message and exit.
    --output -o        : Output format.  Allowed values: json, jsonc, table, tsv.  Default: json.
    --query            : JMESPath query string. See http://jmespath.org/ for more information and
                         examples.
    --verbose          : Increase logging verbosity. Use --debug for full debug logs.
Examples
    List all VMs.
        az vm list
    List all VMs by resource group.
        az vm list -g MyResourceGroup
    List all VMs by resource group with details.
        az vm list -g MyResourceGroup -dt
```

- For example, execute the following command to query for any VM resource within any resource group that contains the letters "My":

```
az vm list --output table --query "[?contains(resourceGroup,'MY')]"
```

Deleting Resources

- The delete command within Azure CLI deletes the resources you no longer need.

```
C:\Users\zdjor> az vm delete -n MyLinuxVM -g MyResourceGroup
Are you sure you want to perform this operation? (y/n): y
EndTime                              Name
StartTime                            Status
-------------------------------  ------------------------------------  ---
-----------------------------
2017-02-19T02:35:56.678905+00:00  5b74ab80-9b29-4329-b483-52b406583e2f
2017-02-19T02:33:35.372769+00:00  Succeeded
```

- We can also use the delete command to delete many resources at a time.
- For example, the following command deletes all the resources in the "`MyResourceGroup`" resource group

```
C:\Users\zdjor> az group delete -n MyResourceGroup
Are you sure you want to perform this operation? (y/n): y
```

Creating other Resources

- We can create many other types of Azure resources as well.
- All new resources are created using a consistent `az <resource type name> create` naming pattern.
- For example, to create an Azure Network Load Balancer (`lb`) that we could then associate with our newly created VMs, we can use the following create command:

```
c:\..> az network lb create -n MyLoadBalancer -g MyResourceGroup
```

- We could also create a new private Virtual Network (commonly referred to as a "VNet" within Azure) for our infrastructure using the following create command:

```
c:\..> az network vnet create -n MyVirtualNetwork -g MyResourceGroup --
address-prefix 10.0.0.0/16
```

- Azure CLI can also create managed platform services. For example, Azure CLI can create an Azure AppService. Azure AppService is a managed platform service that provides a great way to host web apps. Within an AppService, you can create two or more Azure Web Apps using the following create commands

```
az appservice plan create -n MyAppServicePlan -g MyResourceGroup
az webapp create -n MyWebApp43432 -g MyResourceGroup --plan MyAppServicePlan
az webapp create -n MyWebApp43433 -g MyResourceGroup --plan MyAppServicePlan
```

az <resource type name> create pattern

- The following are some popular Azure resource types and the corresponding Azure CLI create commands to create them:

```
Resource Type                  Azure CLI create command
-------------                  ------------------------
Resource Group                 az group create
Virtual Machine                az vm create
Virtual Network                az network vnet create
Load Balancer                  az network lb create
Managed Disk                   az disk create
Storage account                az storage account create
Virtual Machine Scale Set      az vmss create
Azure Container Service        az acs create
Web App                        az webapp create
SQL Database Server            az sql server create
Document DB                    az documentdb create
```

Azure Resource Manager

Resource Manager

- The infrastructure for our applications is typically made up of many components – maybe a virtual machine, storage account, and virtual network, or a web app, database, database server, and 3rd party services.
- We do not see these components as separate entities, instead we see them as related and interdependent parts of a single entity.
- We want to deploy, manage, and monitor those components as a group.
- Azure Resource Manager enables us to work with the resources in our solution as a group.
- We can deploy, update, or delete all the resources for our solution in a single, coordinated operation.
- We use a template for deployment.
- Resource Manager provides security, auditing, and tagging features to help us manage our resources after deployment.

Concepts

- **resource** - A manageable item that is available through Azure. Some common resources are a virtual machine, storage account, web app, database, and virtual network, but there are many more.
- **resource group** - A container that holds related resources for an Azure solution. The resource group can include all the resources for the solution, or only those resources that you want to manage as a group. We decide how you want to allocate resources to resource groups based on what makes the most sense for our organization.
- **resource provider** - A service that supplies the resources you can deploy and manage through Resource Manager. Each resource provider offers operations for working with the resources that are deployed. Some common resource providers are *Microsoft.Compute*, which supplies the virtual machine resource, *Microsoft.Storage*, which supplies the storage account resource, and *Microsoft.Web*, which supplies resources related to web apps.

Concepts

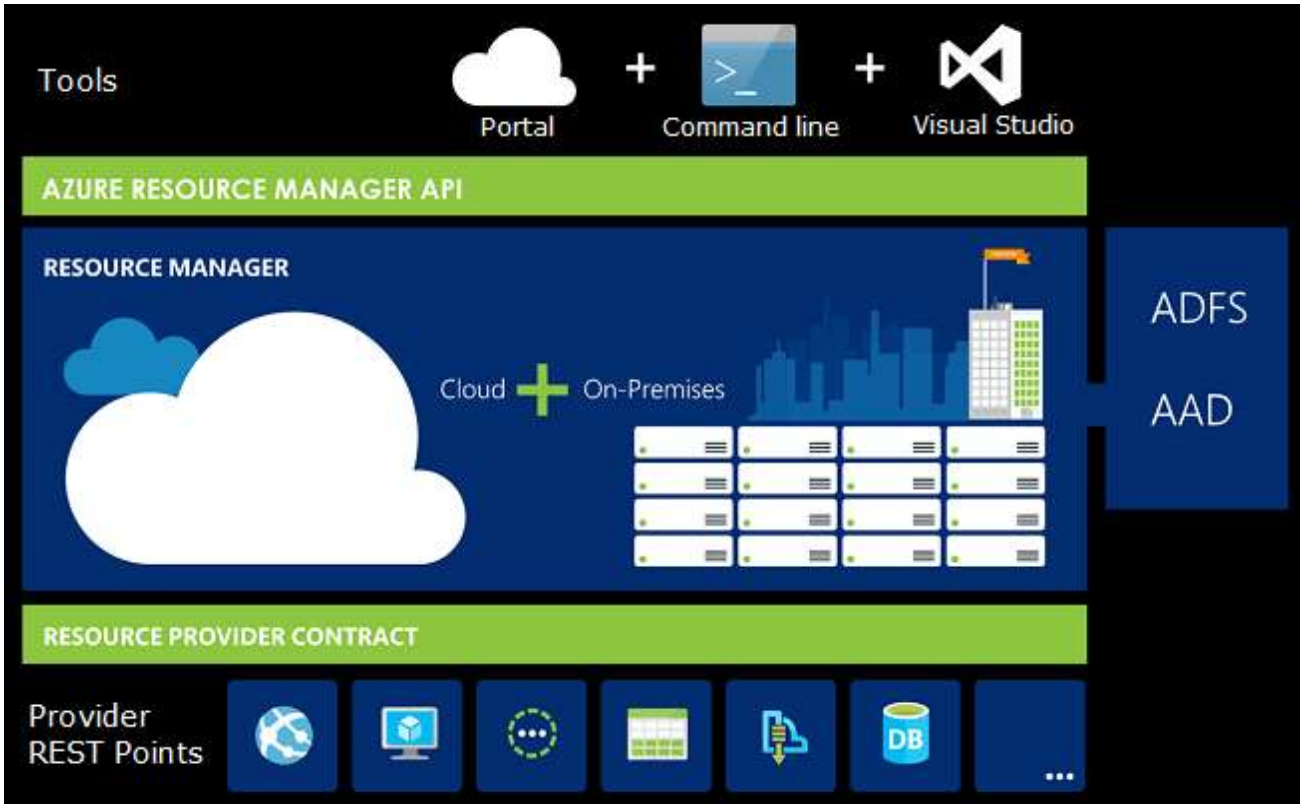
- **Resource Manager template** - A JavaScript Object Notation (JSON) file that defines one or more resources to deploy to a resource group. It also defines the dependencies between the deployed resources. The template can be used to deploy the resources consistently and repeatedly.
- **declarative syntax** - Syntax that lets you state "Here is what I intend to create" without having to write the sequence of programming commands to create it. The Resource Manager template is an example of declarative syntax. In the file, you define the properties for the infrastructure to deploy to Azure.

Benefits of using Resource Manager

- Resource Manager provides several benefits:
- We can deploy, manage, and monitor all the resources for our solution as a group, rather than handling these resources individually.
- We can repeatedly deploy our solution throughout the development lifecycle and have confidence our resources are deployed in a consistent state.
- We can manage our infrastructure through declarative templates rather than scripts.
- We can define the dependencies between resources so they are deployed in the correct order.
- We can apply access control to all services in our resource group because Role-Based Access Control (RBAC) is natively integrated into the management platform.
- We can apply tags to resources to logically organize all the resources in our subscription.
- We can clarify our organization's billing by viewing costs for a group of resources sharing the same tag.

Consistent Management Layer

- Resource Manager provides a consistent management layer for the tasks you perform through Azure PowerShell, Azure CLI, Azure portal, REST API, and development tools. All the tools use a common set of operations.
- We use the tools that work best for us, and can use them interchangeably without confusion.



Recommended Practice

The following suggestions help you take full advantage of Resource Manager when working with your solutions.

- Define and deploy your infrastructure through the declarative syntax in Resource Manager templates, rather than through imperative commands.
- Define all deployment and configuration steps in the template. You should have no manual steps for setting up your solution.
- Run imperative commands to manage your resources, such as to start or stop an app or machine.
- Arrange resources with the same lifecycle in a resource group. Use tags for all other organizing of resources.

Recommendations for Resource Groups

There are some important factors to consider when defining your resource group:

- All the resources in your group should share the same lifecycle. You deploy, update, and delete them together. If one resource, such as a database server, needs to exist on a different deployment cycle it should be in another resource group.
- Each resource can only exist in one resource group.
- You can add or remove a resource to a resource group at any time.
- You can move a resource from one resource group to another group.
- A resource group can contain resources that reside in different regions.
- A resource group can be used to scope access control for administrative actions.
- A resource can interact with resources in other resource groups. This interaction is common when the two resources are related but do not share the same lifecycle (for example, web apps connecting to a database).

Resource Providers

- Each resource provider offers a set of resources and operations for working with an Azure service. For example, if you want to store keys and secrets, you work with the **Microsoft.KeyVault** resource provider. This resource provider offers a resource type called **vaults** for creating the key vault.
- The name of a resource type is in the format: **{resource-provider}/{resource-type}**. For example, the key vault type is **Microsoft.KeyVault/vaults**.
- Before getting started with deploying your resources, you should gain an understanding of the available resource providers.
- Knowing the names of resource providers and resources helps you define resources you want to deploy to Azure.
- Also, you need to know the valid locations and API versions for each resource type.

Resource Providers and Resource Types

- To see all resource providers in Azure, and the registration status for your subscription, use:

```
az provider list --query "[].{Provider:namespace, Status:registrationState}" --out table
Provider                          Status
--------------------------------- ----------------
Microsoft.ClassicCompute          Registered
Microsoft.ClassicNetwork          Registered
Microsoft.ClassicStorage          Registered
Microsoft.CognitiveServices       Registered
```

Registering a Resource Provider

- Registering a resource provider configures your subscription to work with the resource provider. The scope for registration is always the subscription.
- By default, many resource providers are automatically registered. However, you may need to manually register some resource providers.
- To register a resource provider, you must have permission to perform the `/register/action` operation for the resource provider. This operation is included in the Contributor and Owner roles.

```
az provider register --namespace Microsoft.Batch
```

- Which returns a message that registration is on-going.
- You cannot unregister a resource provider when you still have resource types from that resource provider in your subscription.
- To see information for a particular resource provider, use:

```
az provider show --namespace Microsoft.Batch
{
  "id": "/subscriptions/ab293589-2b01-4b26-97a4-
24f36b894fd2/providers/Microsoft.Batch",
  "namespace": "Microsoft.Batch",
  "registrationState": "Registering",
  "resourceTypes": [
    {
      "aliases": null,
      "apiVersions": [ . . . .
```

Find resource types for a provider

- To see the resource types for a resource provider, use:

```
C:\Users\zdjor>az provider show --namespace Microsoft.Batch --query
"resourceTypes[*].resourceType" --out table
Result
-------------------------------
batchAccounts
operations
locations
locations/quotas
locations/checkNameAvailability
```

- The API version corresponds to a version of REST API operations that are released by the resource provider. As a resource provider enables new features, it releases a new version of the REST API. +
- To get the available API versions for a resource type, use:

```
C:\Users\zdjor>az provider show --namespace Microsoft.Batch --query
"resourceTypes[?resourceType=='batchAccounts'].apiVersions | [0]" --out
table
Result
--------------------------
2017-05-01
2017-01-01
2015-12-01
2015-09-01
2015-07-01
2014-05-01-privatepreview
```

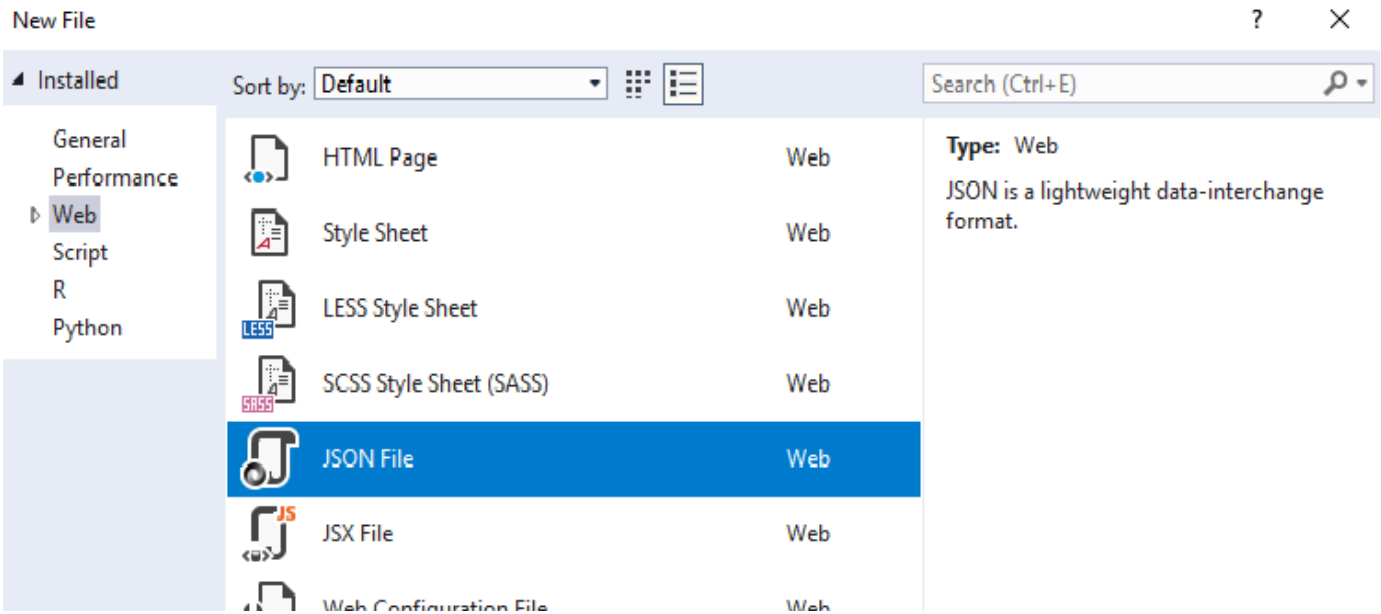
Find regions where type is supported

- To get the supported locations for a resource type, use:

```
C:\Users\zdjor>az provider show --namespace  Microsoft.Batch --query
"resourceTypes[?resourceType=='batchAccounts'].locations | [0]" --out table
Result
-------------------
West Europe
East US
East US 2
West US
North Central US
Brazil South
North Europe
Central US
East Asia
Japan East
```

Template Example

- In Visual Studio, select File > New > File
- On New File page, select Web and then JSON



- On the bottom of the page select Open
- In the white editor space that opens paste text on the following slide.
- Storage account names have several restrictions that make them difficult to set. The name must be between 3 and 24 characters in length, use only numbers and lower-case letters, and be unique. The following template uses the `uniqueString()` function to generate a hash value.
- To give this hash value more meaning, it adds the prefix *storage*.

Example template text

```
{
    "$schema": "http://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
    },
    "variables": {
    },
    "resources": [
        {
            "name": "[concat('storage', uniqueString(resourceGroup().id))]",
            "type": "Microsoft.Storage/storageAccounts",
            "apiVersion": "2016-01-01",
            "sku": {
                "name": "Standard_LRS"
            },
            "kind": "Storage",
            "location": "South Central US",
            "tags": {},
            "properties": {}
        }
    ],
    "outputs": {}
}
```

- Save this file with name `mydeploy.json` to a local folder

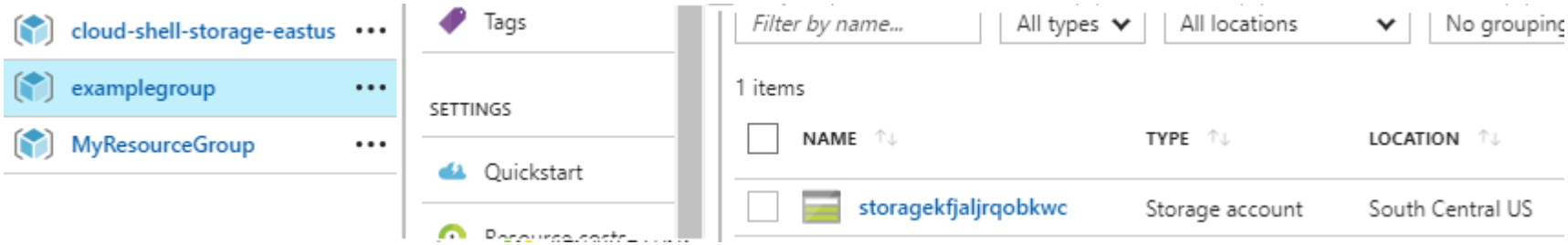
Deploy template

```
C:\Users\zdjor>az group deployment create --resource-group examplegroup --template-file mydeploy.json
{| Finished ..
  "id": "/subscriptions/ab293589-2b01-4b26-97a4-
24f36b894fd2/resourceGroups/examplegroup/providers/Microsoft.Resources/deployments/mydeploy",
  "name": "mydeploy",
  "properties": {
    "correlationId": "8ea76f2e-8418-4206-82da-c6c180797496", "debugSetting": null, "dependencies": [],
    "mode": "Incremental",
    "outputs": {},     "parameters": {},
    "parametersLink": null,
    "providers": [
      {
        "id": null, "namespace": "Microsoft.Storage",
        "registrationState": null,
        "resourceTypes": [
          {
            "aliases": null, "apiVersions": null,
            "locations": [
              "southcentralus"
            ],
            "properties": null, "resourceType": "storageAccounts"
          }
        ]
      }
    ],
    "provisioningState": "Succeeded",
    "template": null,
    "templateLink": null,
    "timestamp": "2017-10-31T20:45:21.420243+00:00"
  },
  "resourceGroup": "examplegroup"
}
```

Verify Deployment

```
C:\Users\zdjor>az group list --out table
Name                          Location        Status
----------------------------  --------------  ---------
cloud-shell-storage-eastus    eastus          Succeeded
examplegroup                  southcentralus  Succeeded
MyResourceGroup               southcentralus  Succeeded
```

- In Portal:



- You want your templates to be more flexible and various names and variables to be passed to them from the command line or programmatically.

Parametrized Template

- The example shows the parameters section with two parameters. The first parameter storageSKU enables you to specify the type of redundancy. It limits the values you can pass in to values that are valid for a storage account. It also specifies a default value. The second parameter storageNamePrefix is set to allow a maximum of 11 characters. It specifies a default value.

```
"parameters": {
  "storageSKU": {
    "type": "string",
    "allowedValues": [
      "Standard_LRS",
      "Standard_ZRS",
      "Standard_GRS",
      "Standard_RAGRS",
      "Premium_LRS"
    ],
    "defaultValue": "Standard_LRS",
    "metadata": {
      "description": "The type of replication
to use for the storage account."
    }
  },
  "storageNamePrefix": {
    "type": "string",
    "maxLength": 11,
    "defaultValue": "storage",
    "metadata": {
      "description": "The value to use for
starting the storage account name. Use only
lowercase letters and numbers."
    }
  }
},
```

Variables and Resources

- In the variables section, add a variable named `storageName`. It combines the prefix value from the parameters and a hash value from the `uniqueString` function. It uses the `toLower` function to convert all characters to lowercase.

```
"variables": {
  "storageName": "[concat(toLower(parameters('storageNamePrefix')),
uniqueString(resourceGroup().id))]"
},
```

- To use these new values for your storage account, change the `resources` definition:

```
"resources": [
  {
    "name": "[variables('storageName')]",
    "type": "Microsoft.Storage/storageAccounts",
    "apiVersion": "2016-01-01",
    "sku": {
      "name": "[parameters('storageSKU')]"
    },
    "kind": "Storage",
    "location": "[resourceGroup().location]",
    "tags": {},
    "properties": {}
  }
],
```

- The name of the storage account is now set to the variable that you added.
- The SKU name is set to the value of the parameter.
- The location is set the same location as the resource group.

Modified Template

```
{ "$schema": "http://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "storageSKU": {
      "type": "string",
      "allowedValues": [
        "Standard_LRS", "Standard_ZRS", "Standard_GRS", "Standard_RAGRS", "Premium_LRS"
      ], "defaultValue": "Standard_LRS",
      "metadata": { "description": "The type of replication to use for storage account." }
    },
    "storageNamePrefix": {
      "type": "string", "maxLength": 11,  "defaultValue": "storage",
      "metadata": {   "description": "Use for starting the storage account name."
      }     }   },
  "variables": {  "storageName": "[concat(toLower(parameters('storageNamePrefix')),
uniqueString(resourceGroup().id))]"
  },   "resources": [  {
      "name": "[variables('storageName')]", "type": "Microsoft.Storage/storageAccounts",
      "apiVersion": "2016-01-01",
      "sku": {
        "name": "[parameters('storageSKU')]"
      },
      "kind": "Storage","location":"[resourceGroup().location]", "tags": {}, "properties": {}
    }   ],
  "outputs": {  } }
```

Redeploy Template

```
C:\Users\zdjor> az group deployment create --resource-group examplegroup --
template-file mydeploy2.json --parameters storageSKU=Standard_RAGRS
storageNamePrefix=newstore
 \ Running ...
{| Finished ..
  "id": "/subscriptions/ab293589-2b01-4b26-97a4-
24f36b894fd2/resourceGroups/examplegroup/providers/Microsoft.Resources/deployment
s/mydeploy2",
  "name": "mydeploy2",
  "properties": {
    "correlationId": "5cf9713b-00ae-48cd-91f2-966fa4b9ee1b",
    "debugSetting": null,
    "dependencies": [],
    "mode": "Incremental",
    "outputs": {},
    "parameters": {
      "storageNamePrefix": {
        "type": "String",
        "value": "newstore"
      },
      "storageSKU": {
        "type": "String",
        "value": "Standard_RAGRS"
      }
    },
```

Redeploy Template


```
 "parametersLink": null,
    "providers": [
      {
        "id": null,
        "namespace": "Microsoft.Storage",
        "registrationState": null,
        "resourceTypes": [
          {
            "aliases": null,
            "apiVersions": null,
            "locations": [
              "southcentralus"
            ],
            "properties": null,
            "resourceType": "storageAccounts"
          }
        ]
      }
    ],
    "provisioningState": "Succeeded",
    "template": null,
    "templateLink": null,
    "timestamp": "2017-10-31T21:15:14.800225+00:00"
  },
  "resourceGroup": "examplegroup"
}

C:\Users\zdjor>
```

Microservices and

Microservices

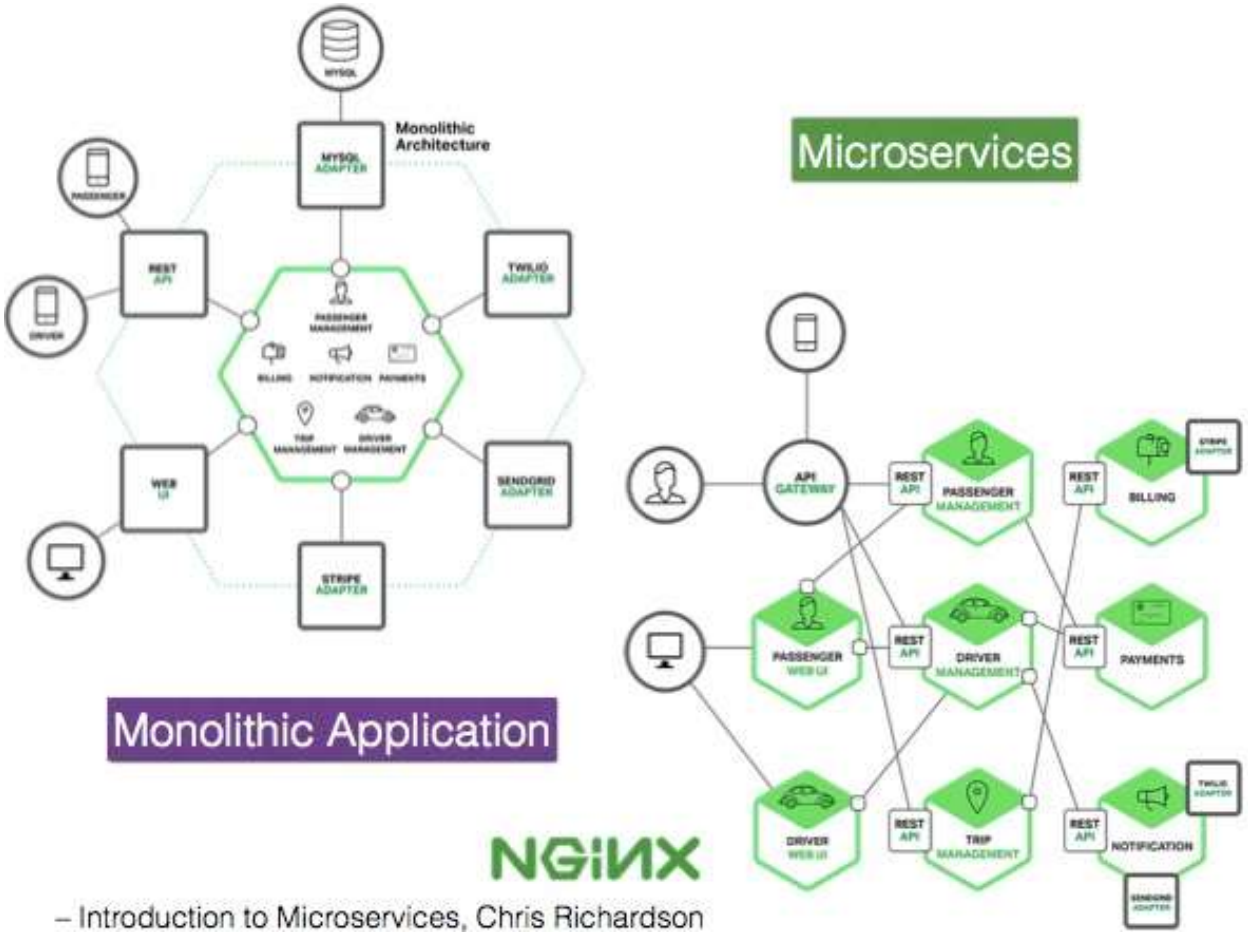
- **Microservices** is a new architectural approach used by large institutions like Amazon and Google.
- Microservice Architecture (MSA) is an evolution of service-oriented architecture (SOA) used to build flexible, independently deployable systems.
- Services in a MSA are processes that communicate with each other over a network. These services communicate thru technology-agnostic protocols.
- MSA is a first realization of SOA that followed the introduction of DevOps and is popular for building continuously deployable systems.
- In MSA, services should have a small granularity and the protocols should be lightweight. All services should be independently deployable.
- Distribution of different responsibilities into separate smaller services enhances the cohesion and decreases the coupling. This makes it easier to change and add functions and qualities to the system at any time.
- In MSA individual service could be continuously refactored what eliminates big up-front designs and allows releasing software early and continuously.

MSA Characteristics

- Some of the defining characteristics of MSA include:
- The services are easy to replace.
- Services are organized around capabilities, e.g., user interface front-end, recommendation, logistics, billing, etc.
- Services can be implemented using different programming languages, databases, hardware and software environment, depending on what fits best.
- Services are small in size, messaging enabled, bounded by contexts, autonomously developed, independently deployable, decentralized and built and released with automated processes.
- All structures are modular
- MSA became a synonym for continuous delivery software development process. A change to a small part of the application only requires one or a small number of services to be rebuilt and redeployed.[2]

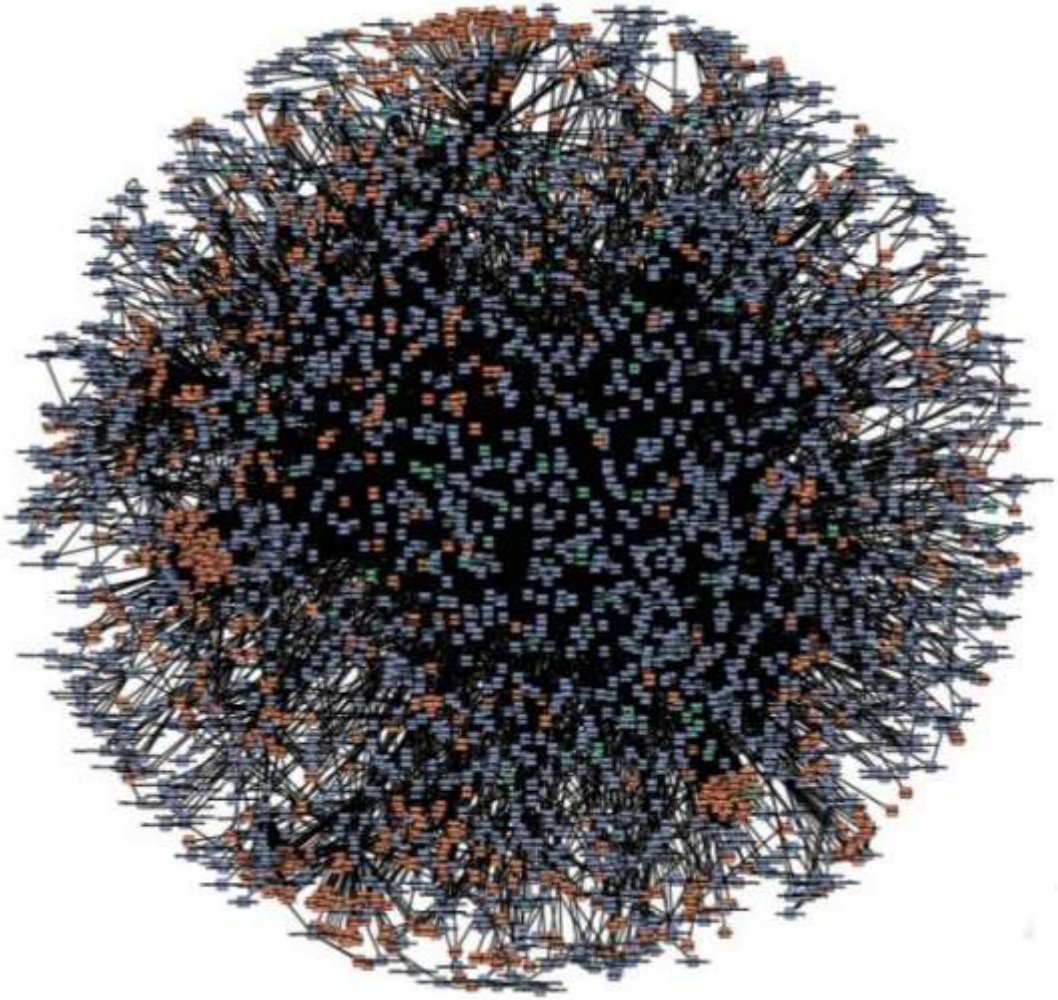
MSA Architectural Diagrams

- Microservices implementations range in complexity:



MSA Architectural Diagrams

- Microservices implementations range in complexity. This is an image of AWS MSA:



- At AWS each service is developed and maintained by a 2 pizza team.

DOCKER, Some History

- Docker and similar container technologies are key ingredients for successful Microservices implementation.
- Docker is an open platform for developers and sysadmins to build, ship and run distributed applications
- A dotCloud (PAAS provider) project
- Initial commit January 18, 2013
- Docker 0.1.0 released March 25, 2013
- Can run on popular 64-bit Linux distributions with kernel 3.8 or later
- Can run on Windows 10 Professional and Enterprise, WIN Server 2016
- Supported by several cloud platforms including Amazon EC2, Google Cloud Platform and Azure

www.docker.com

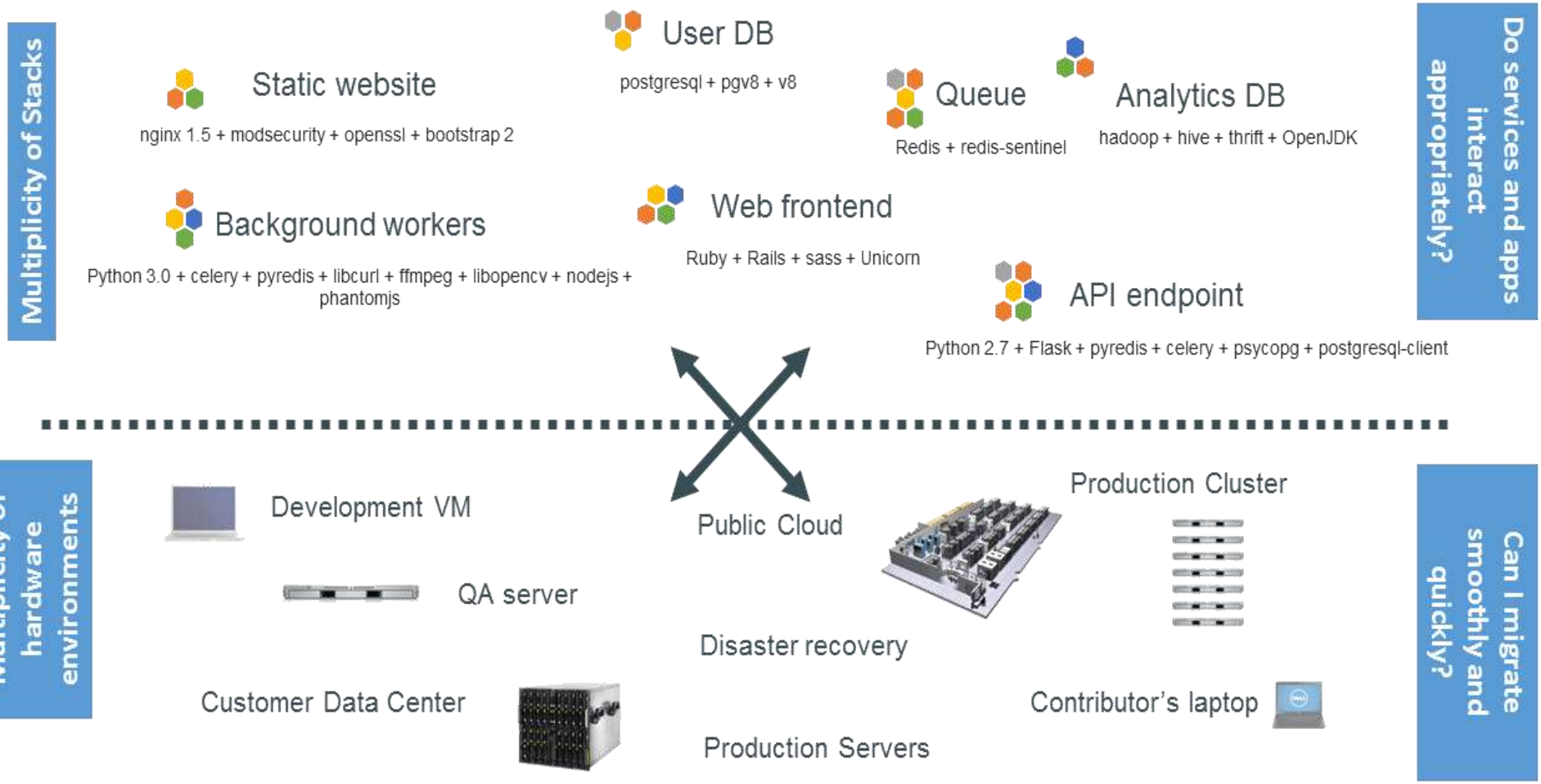
What does Docker, Inc. do?

- Builds Docker Engine - open source container management.
- Maintains Docker Hub - online home and hub for managing your Docker containers.
- Provides Docker Enterprise Support - commercial support for Docker.
- Provides Docker Services & Training - professional services and training to help you get the best out of Docker.

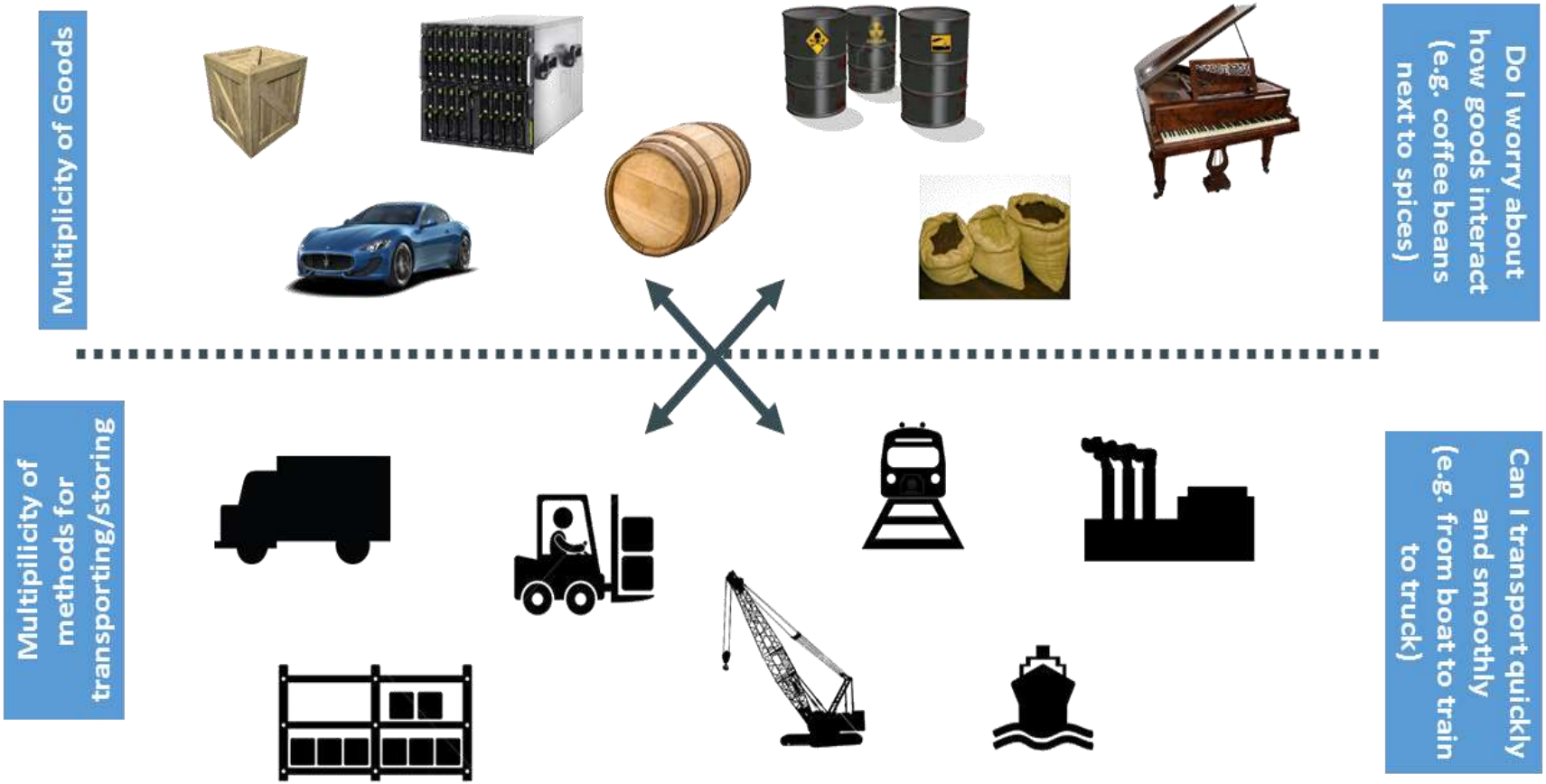
Features

- Light-Weight
  - Minimal overhead (*cpu/io/network*)
  - Based on Linux containers
  - Uses layered filesystem to save space (AUFS/LVM)
  - Uses a copy-on-write filesystem to track changes
- Portable
  - Can run on almost any Linux system.
  - Raspberry pi support.
  - Windows
- Self-sufficient
  - A Docker container contains everything it needs to run
  - Minimal Base OS, Libraries and frameworks
  - Application code
  - A Docker container should be able to run anywhere that Docker can run (most Linux distributions and new Windows.

The Challenge



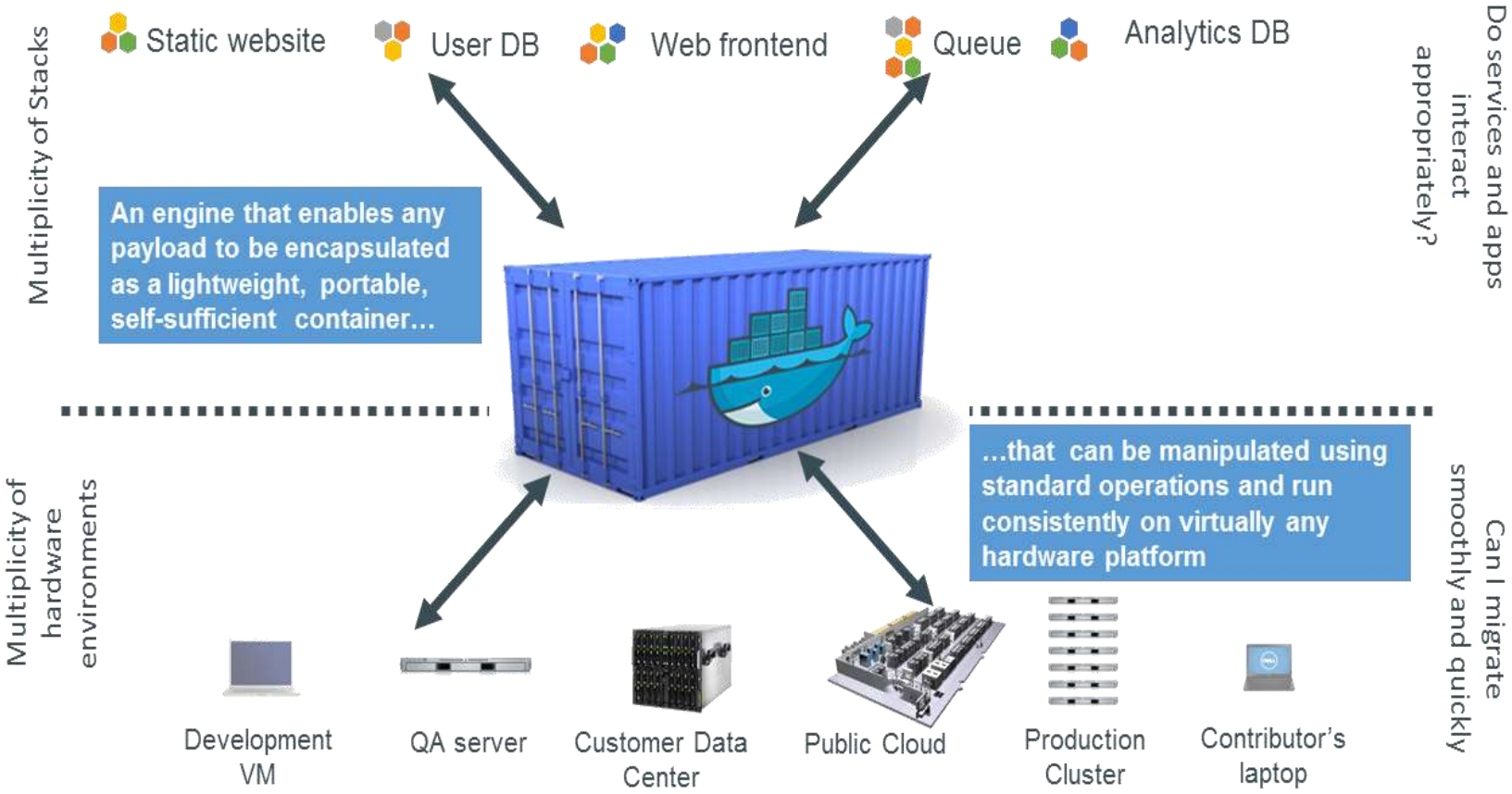
Cargo Transport Pre-1960



Solution: Intermodal Shipping Container

- 90% of trade today is shipped in containers.
- Order of magnitude reduction in costs.
- Made possible globalization. Fruits from Chile and everything we buy that is Made in China.

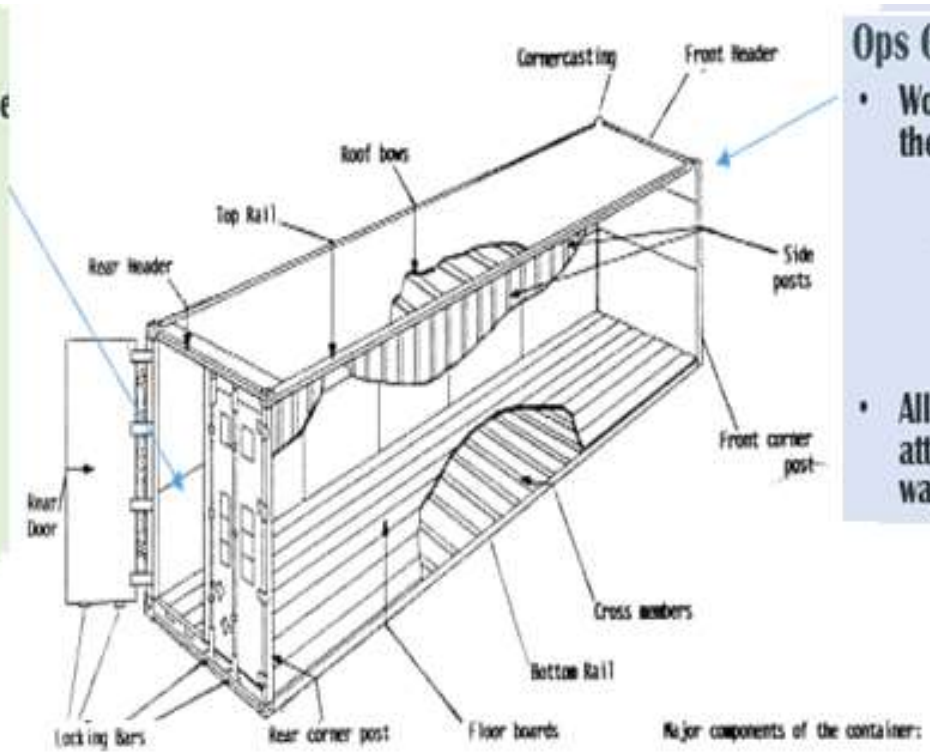
Docker is a Container System for Code



Why it Works: Separation of Concerns

Developer

- Worries about what's "inside" the container
  - His code
  - His Libraries
  - His Package Manager
  - His Apps
  - His Data
- All Linux servers look the same

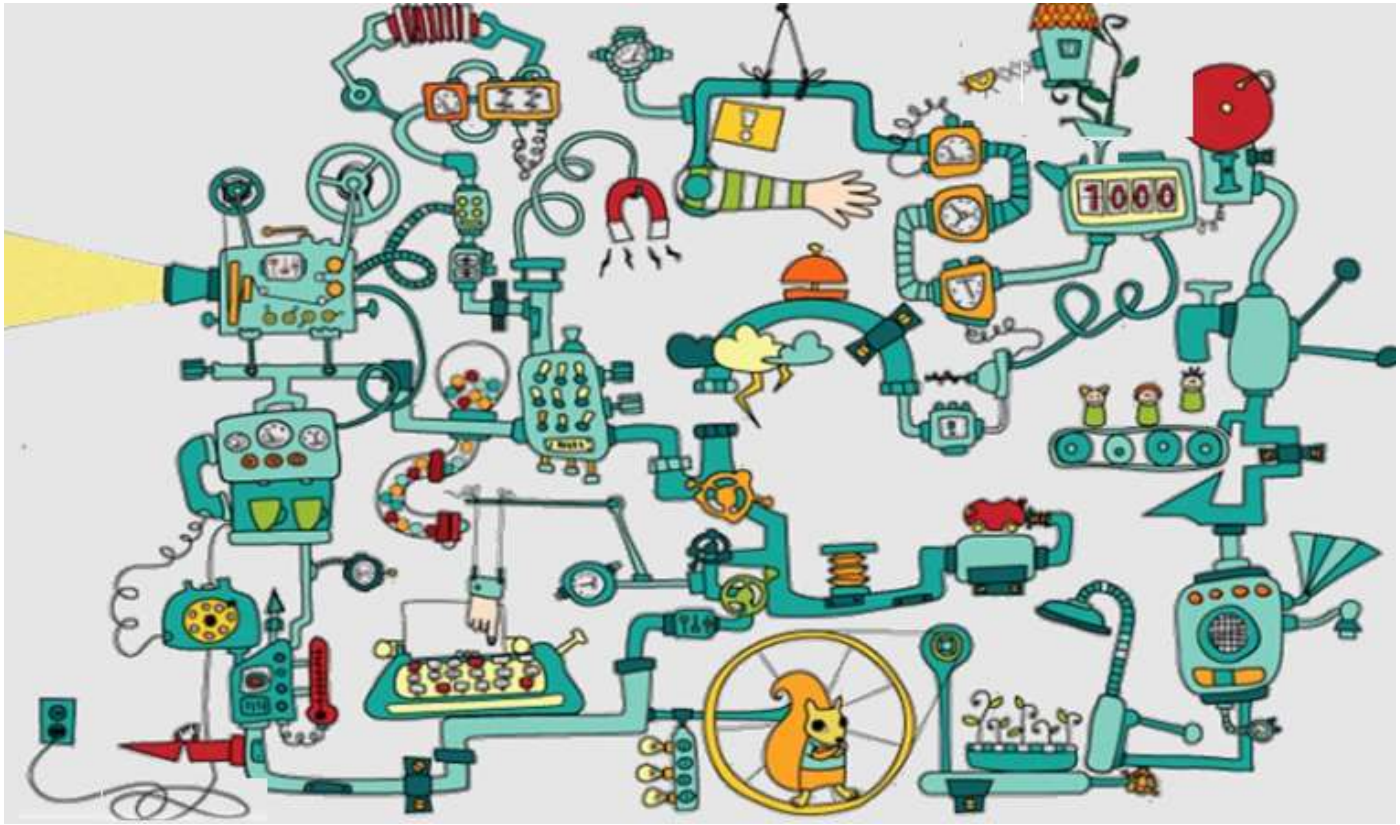


Ops Guy

- Worries about what's "outside" the container
  - Logging
  - Remote access
  - Monitoring
  - Network config
- All containers start, stop, copy, attach, migrate, etc. the same way

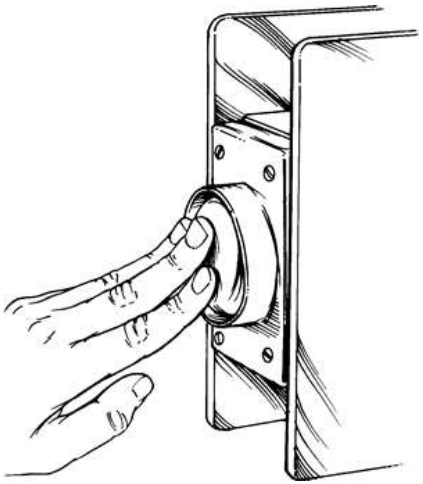
- The greatest benefit of containers is complete separation of concerns.
- Developers stuff a container with all libraries and code they need.
- Operations team pushes all containers into production environment in the same way.
- All containers are the same from the deployment perspective.

Containers existed before Docker



- Containers have been around for a while (c.f. LXC, Solaris Zones, BSD, Jails)
- Their use was largely limited to specialized organizations, with special tools & training. Containers were not portable

Containers after Docker



Obviously, Docker is yet another best thing after the

With Docker, Containers get the following:

- Ease of use, tooling, Open Stack
- Re-usable components
- Ability to run on any Linux server today: physical, virtual, VM, cloud,
- Ability to move between any of the above in a matter of seconds-no modification or delay
- Ability to share containerized components
- Self contained environment - no dependency hell
- Tools for containers work together: linking, nesting, discovery orchestration

Docker Container

Container is a unit of software delivery

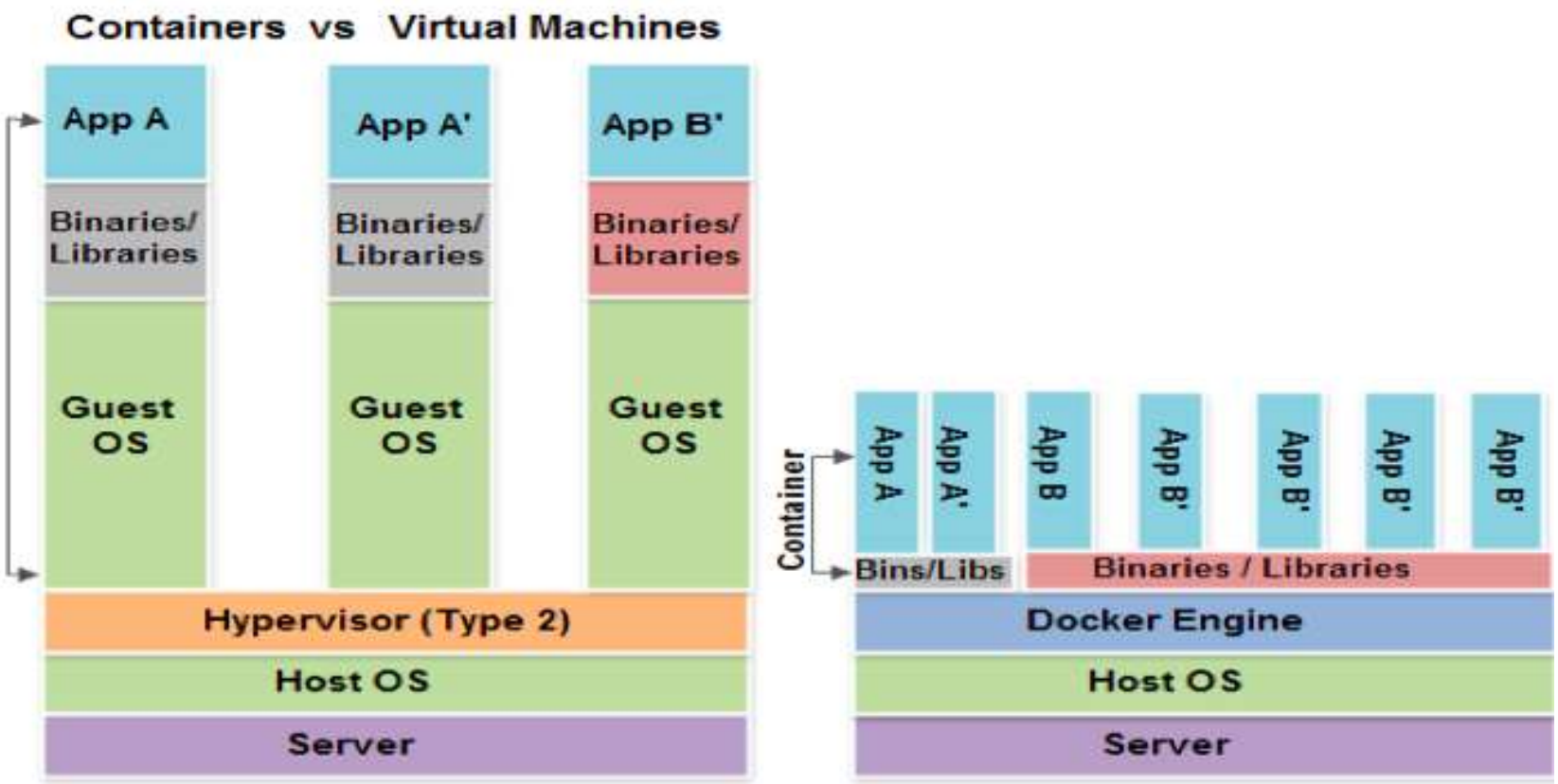
- runs everywhere
  - regardless of kernel version
  - regardless of host distribution
  - container and host architecture must match

- runs anything
  - if it can run on the host, it can run in the container
  - if it can run on a Linux kernel, it can run

How does Docker work

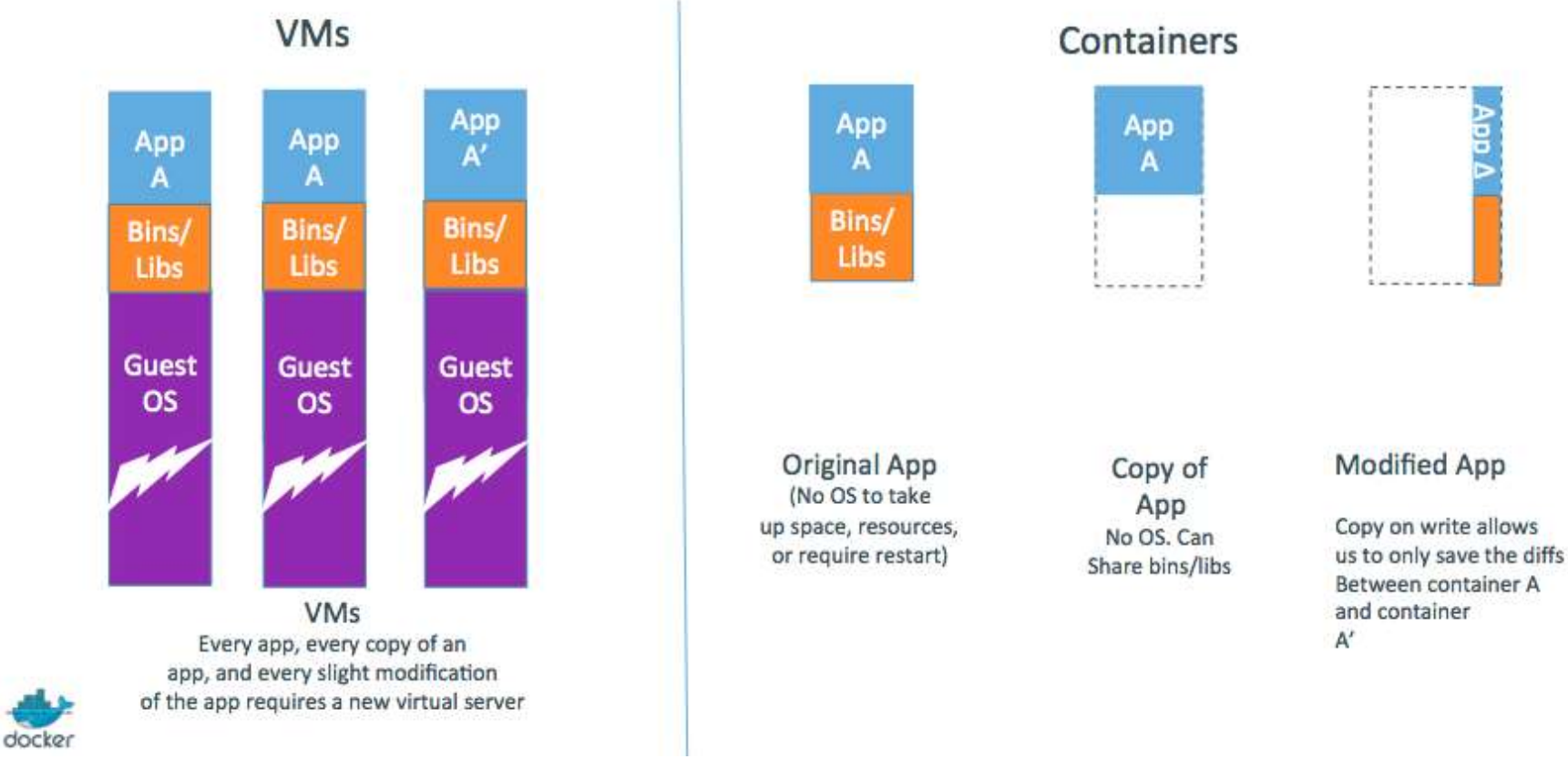
- You build Docker images that hold your applications
- You create Docker containers from those Docker images to run your applications.
- You can share those Docker images via Docker Hub or your own registry

Virtual Machine Versus Container



- Docker containers are much lighter than virtual machines. They rely on the most basic features of the Linux OS.
- Containers are completely isolated but different containers could share libraries.

Difference between VMs and Containers



Docker Container Lifecycle

The Life Stages of a Container

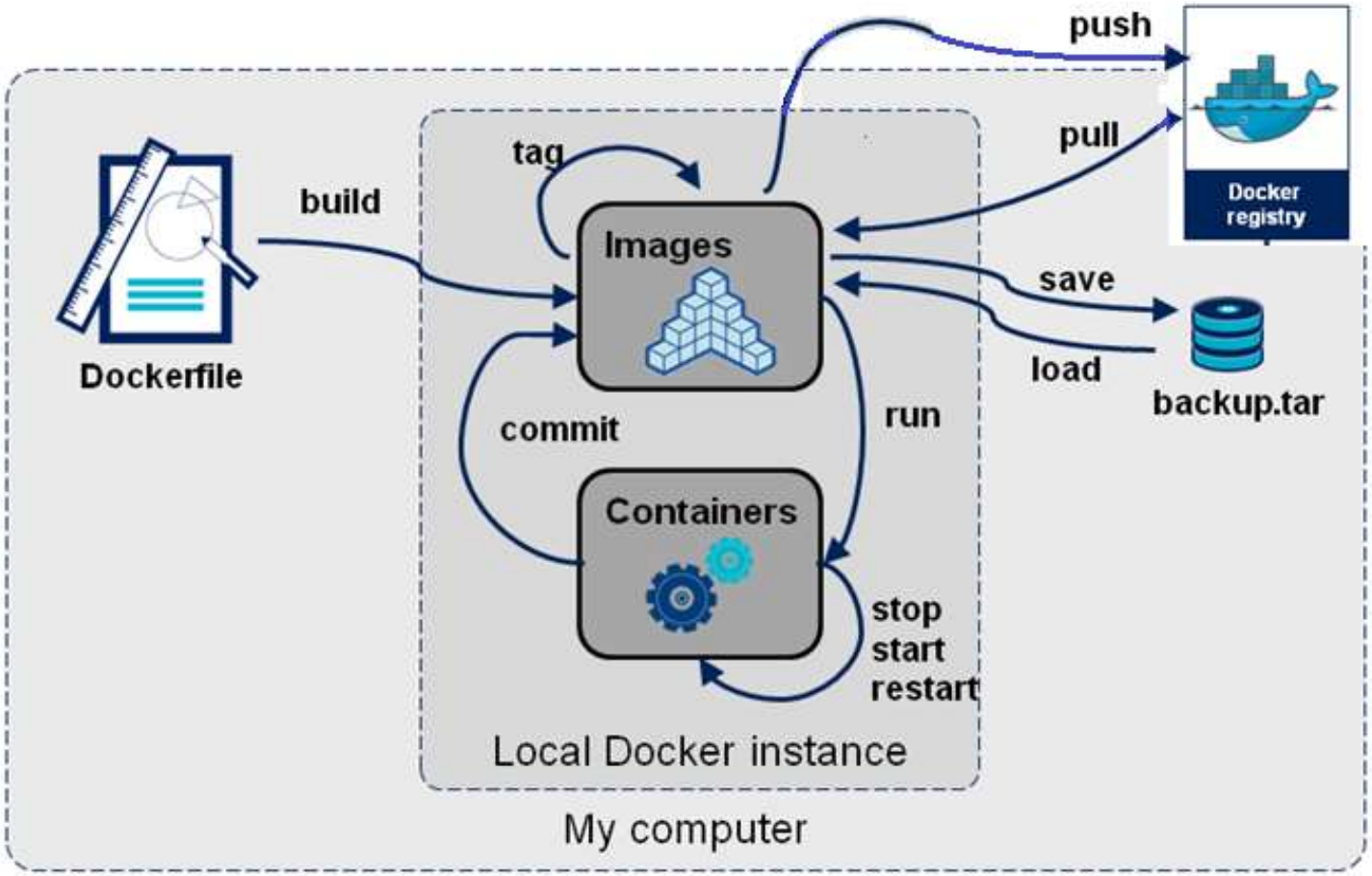
- Conception
  - **BUILD** an Image from a Dockerfile
- Birth
  - **RUN** (create+start) a container
- Reproduction
  - **COMMIT** (persist) a container to a new image
  - **RUN** a new container from an image
- Sleep
  - **KILL** a running container
- Wake
  - **START** a stopped container
- Death
  - **RM** (delete) a stopped container

-- Extinction

RMI a container image (delete image)

Docker Lifecycle and Processes

- Docker Repository is public like DockerHub or private, in the Cloud or in your Organization.



Where to get Docker for Windows

- If you run Windows 10 Professional or Enterprise 64 bit, you can download Docker for Windows at

https://www.docker.com/docker-windows

- If you have older Windows, try Docker Toolbox at :

https://www.docker.com/products/docker-toolbox

- In many ways you are better of with a Linux VM, either Ubuntu or CentOS. Docker appears to be better behaved on those systems.

Overview

Available for both Windows and Mac, the Toolbox installs Docker Client, Machine, Compose and Kitematic.

What's in the Toolbox

- DOCKER ENGINE
- COMPOSE
- MACHINE
- KITEMATIC

Install on Your Linux VM

- Docker people prefer Ubuntu Linux.
- Build your VM with (almost) any Linux IS.
- On Red Hat and derivatives, CentOS, Fedora, others

```
$ sudo yum install docker
```

- On Debian/Ubuntu and derivatives.

```
$ sudo apt-get update
$ sudo apt-get install docker.io
```

- You can use the `curl` command for installation on several platforms.

```
$ curl -s https://get.docker.io/centos/ | sudo sh
```

- This currently works on:
- Ubuntu; Debian; Fedora; Gentoo.
- Installation on older Mac OsX and Windows requires installation of a special VM.
- For Mac instructions go to http://docs.docker.com/mac/started/
- To start docker engine type

```
$ sudo service docker start
```

Docker and UFW on Ubuntu

- If you use the UFW, or Uncomplicated Firewall, on Ubuntu, then you'll need to make a small change to get it to work with Docker.
- Docker uses a network bridge to manage the networking on your containers. By default, UFW drops all forwarded packets. You'll need to enable forwarding in UFW for Docker to function correctly. We can do this by editing the `/etc/default/ufw` file. Inside this file, change:
- **Old UFW forwarding policy** `DEFAULT_FORWARD_POLICY="DROP"` To:
- **New UFW forwarding policy** `DEFAULT_FORWARD_POLICY="ACCEPT"`
- Save the update, enable and reload UFW.

```
$ sudo ufw enable
$ sudo ufw reload
```

- If you are testing and developing you may as well leave UFW disabled.

curl, git

- `curl and git` are very useful utilities. If you do not have them installed,
- On Ubuntu type:

```
$ apt-get install curl
$ apt-get install git
```

- On Red Hat like systems (CentOS, Fedora), type:

```
$ yum install curl
$ yum install git
```

Alternative, use `wget`

- Verify that you have `wget` installed.

`$ which wget` If `wget` isn't installed, install it after updating your manager:

```
$ sudo apt-get update
$ sudo apt-get install wget
```

- Get the latest Docker package.

```
$ wget -qO- https://get.docker.com/ | sh
```

- The system prompts you for your sudo password. Then, it downloads and installs Docker and its dependencies.
- Note: If your company is behind a filtering proxy, you may find that the apt-key command fails for the Docker repo during installation. To work around this, add the key directly using the following:

```
$ wget -qO- https://get.docker.com/gpg | sudo apt-key add -
```

- Verify `docker` is installed correctly. Type:

```
$ docker run hello-world
```

Installation on CentOS 7, from www.docker.com

- Docker is supported on CentOS 7.X
- Installation on other binary compatible EL7 distributions such as Scientific Linux might succeed, but Docker.com does not test or support Docker on these distributions.
- Docker requires a 64-bit installation regardless of your CentOS version. Also, your kernel must be 3.10 at minimum, which CentOS 7 runs
- To check your current kernel version, open a terminal and use `uname -r` to display your kernel version:

```
$ uname -r
3.10.0-693.2.2.el7.x86_64
```

- Finally, is it recommended that you fully update your system. Please keep in mind that your system should be fully patched to fix any potential kernel bugs. Update all packages

```
$ sudo yum update          # then run installation script
$ curl -sSL https://get.docker.com/ | sh
```

Open Firewall on CentOS host

- If you know that your container will for example have a running Web server and you want to access that server at port 8080, you need to modify local firewall on CentOS host (VM) to allow port 8080 through.
- Type:

```
$ sudo firewall-cmd --permanent --add-port=8080/tcp and
$ sudo firewall-cmd --reload
```

Install EPEL on older Red Hat, CentOS, Fedora

- Docker might work on older CentOS and Red Hats with kernel 2.6 and newer. On those systems you have to install EPEL by adding the following RPM. Type all on one line:

```
$ sudo rpm -Uvh
http://download.fedoraproject.org/pub/epel/6/i386 /epel-
release-6-8.noarch.rpm
```

- Afterward, on older Linux machine, you should be able to install the Docker package.

```
$ sudo yum -y install lxc-docker
```

- On new CentOS machines the following should work

```
$ sudo yum -y install docker
```

Starting the Docker daemon on Red Hat family

- Once the package is installed, we can start the Docker daemon. On Red Hat Enterprise Linux 6 and CentOS 6 you can use.

```
$ sudo service start docker
```

- If we want Docker to start at boot we should also:

```
$ sudo service enable docker
```

- On Red Hat Enterprise 7, CentOS 7 and latest Fedora to **start the Docker service and configure it to start at the boot time**, type

```
$ sudo systemctl enable docker.service    # and
$ sudo systemctl start docker.service
```

- Startup at the boot time is enabled by the first command:

```
$ sudo systemctl enable docker.service
```

Test whether Docker is working

- Using the `docker` client:

```
$ sudo docker version
Client:
 Version:         1.12.6
 API version:     1.24
 Package version: docker-1.12.6-61.git85d7426.el7.centos.x86_64
 Go version:      go1.8.3
 Git commit:      85d7426/1.12.6
 Built:           Tue Oct 24 15:40:21 2017
 OS/Arch:         linux/amd64
Server:
 Version:         1.12.6
 API version:     1.24
 Package version: docker-1.12.6-61.git85d7426.el7.centos.x86_64
 Go version:      go1.8.3
 Git commit:      85d7426/1.12.6
 Built:           Tue Oct 24 15:40:21 2017
 OS/Arch:         linux/amd64
```

Test whether Docker is working

- Using `ps -ef`

```
[centos@localhost ~]$ ps -ef | grep docker
root       4266      1  2 13:19 ?        00:00:00
/usr/bin/docker daemon --selinux-enabled
```

- Run Hello World like container `busybox`

```
$ sudo docker run busybox echo hello world
hello world
```

Create `docker` Linux group,

- The `docker` or `dockerroot` user is `root` equivalent. It provides `root` level access.
- You should restrict access to it like you would protect root. Add the `docker` group if it is not there already.

```
$ sudo groupadd docker
```

- Add user `centos ($USER)` to the group

```
$ sudo gpasswd -a $USER dockerroot
```

- Restart the Docker daemon

```
$ sudo systemctl restart docker.service
```

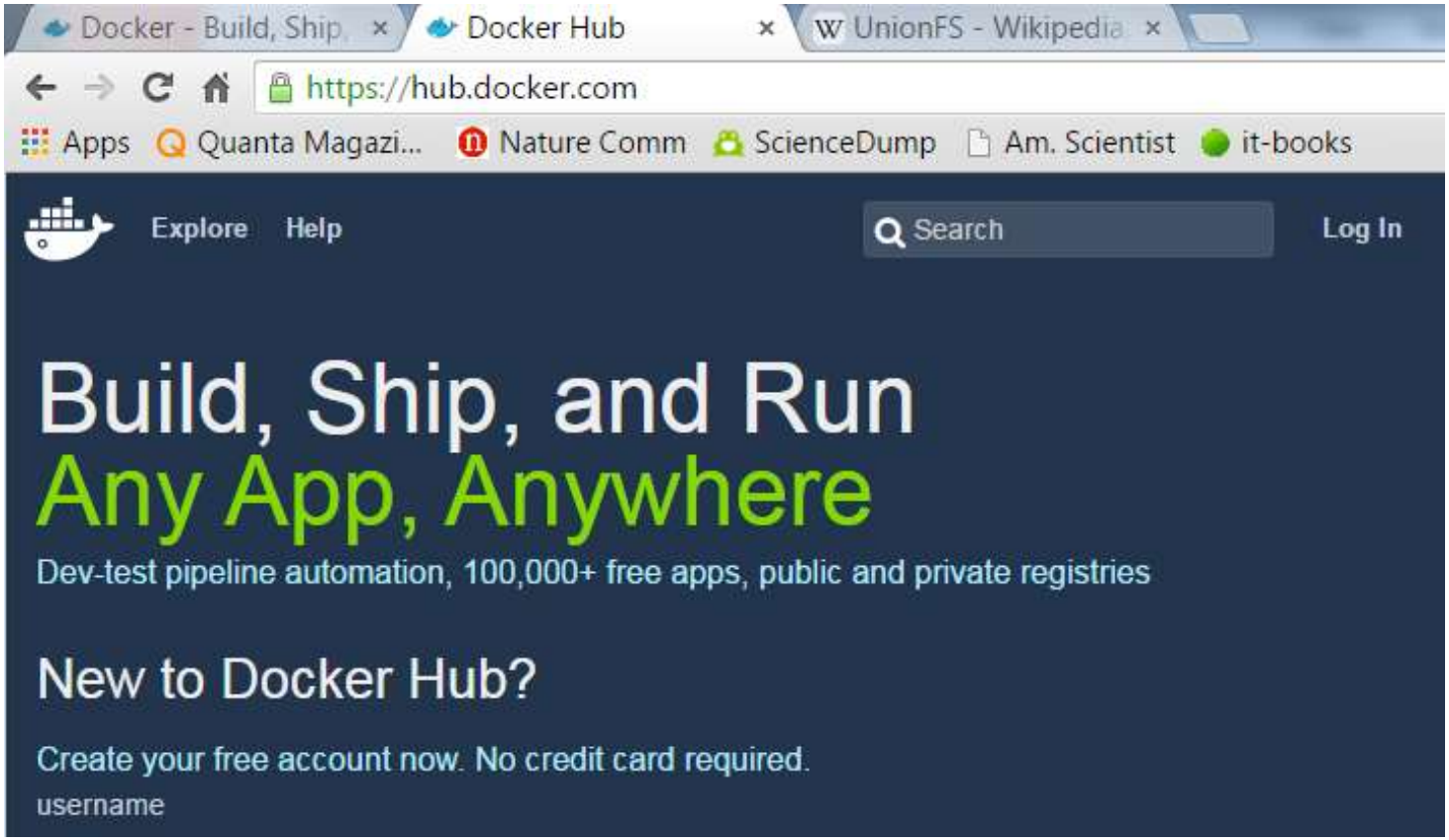
- You may, as well, be `root` all the time. Your main VM user (e.g. `centos`) has `sudo` privileges so you could do:

```
$ sudo su
```

- And be `root` forever.

Docker Hub

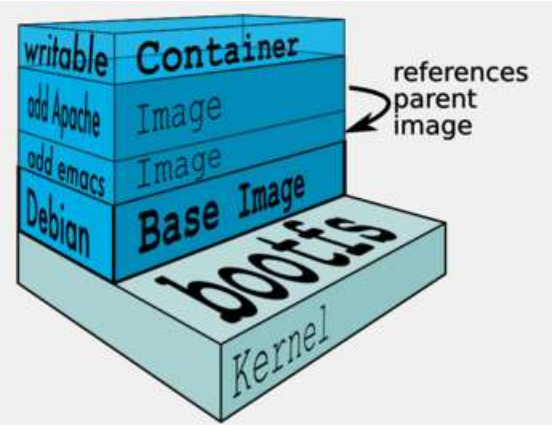
- Docker Hub account will allow us to store our images in the registry.
- To sign up, you'll go to `hub.docker.com` and fill out the form.
- Activate Docker Hub account through email and a confirmation link.



Images

- An image is a collection of files.
- *Base images* (ubuntu, busybox, fedora etc.) are what you build your own custom images on top of.
- Images are *layered*, and each layer represents a diff (what changed) from the previous layer. For instance, you could add Python 3 on top of a base image.

- Images can be stored:
  - On your Docker host.
  - In a Docker registry.
- You can use the Docker client to manage images.



Search for Images

- Searches your registry for images:

```
$ docker search training
NAME                    DESCRIPTION STARS OFFICIAL AUTOMATED
training/jenkins                     0               [OK]
training/webapp                      0               [OK]
training/ls                          0               [OK]
training/namer                       0               [OK]
training/postgres                    0               [OK]
training/notes                       0               [OK]
```

- Images belong to a namespace. There are several namespaces:
- Root-like
  - ubuntu
- User
  - training/docker-fundamentals-image
- Self-Hosted
  - registry.example.com:5000/my-private-image

Containers vs. Images

- Containers represent an encapsulated set of processes based on an image.
- You spawn them with the `docker run` command.
- In our previous example, you created a shiny new container by executing `docker run`. It was based on the `busybox` image, and we ran the `echo` command.
- Images are like templates or stencils that you can create containers from.

docker --help, CLI Commands

```
zdjordje@localhost build]$ sudo docker --help
Usage: docker [OPTIONS] COMMAND [arg...]
       docker [ --help | -v | --version ]

A self-sufficient runtime for containers.
Options:
  --config=~/.docker              Location of client config files
  -D, --debug                     Enable debug mode
  -H, --host=[]                   Daemon socket(s) to connect to
  -h, --help                      Print usage
  -l, --log-level=info            Set the logging level
  --tls                           Use TLS; implied by --tlsverify
  --tlscacert=~/.docker/ca.pem    Trust certs signed only by this CA
  --tlscert=~/.docker/cert.pem    Path to TLS certificate file
  --tlskey=~/.docker/key.pem      Path to TLS key file
  --tlsverify                     Use TLS and verify the remote
  -v, --version                   Print version information and quit
Commands:
    attach    Attach to a running container
    build     Build an image from a Dockerfile
    commit    Create a new image from a container's changes
    cp        Copy files/folders between a container and the local filesystem
    create    Create a new container
    diff      Inspect changes on a container's filesystem
    events    Get real time events from the server
```

docker -help, CLI Commands

```
exec       Run a command in a running container
export     Export a container's filesystem as a tar archive
history    Show the history of an image
images     List images
import     Import the contents from a tarball to create a filesystem image
info       Display system-wide information
inspect    Return low-level information on a container, image or task
kill       Kill one or more running containers
load       Load an image from a tar archive or STDIN
login      Log in to a Docker registry.
logout     Log out from a Docker registry.
logs       Fetch the logs of a container
network    Manage Docker networks
node       Manage Docker Swarm nodes
pause      Pause all processes within one or more containers
port       List port mappings or a specific mapping for the container
ps         List containers
pull       Pull an image or a repository from a registry
push       Push an image or a repository to a registry
rename     Rename a container
restart    Restart a container
rm         Remove one or more containers
rmi        Remove one or more images
run        Run a command in a new container
```

docker -help, CLI Commands

```
    exec      Run a command in a running container
    save      Save one or more images to a tar archive (streamed to STDOUT by
default)
    search    Search the Docker Hub for images
    service   Manage Docker services
    start     Start one or more stopped containers
    stats     Display a live stream of container(s) resource usage statistics
    stop      Stop one or more running containers
    swarm     Manage Docker Swarm
    tag       Tag an image into a repository
    top       Display the running processes of a container
    unpause   Unpause all processes within one or more containers
    update    Update configuration of one or more containers
    version   Show the Docker version information
    volume    Manage Docker volumes
    wait      Block until a container stops, then print its exit code
```

Downloading Images

- Download a user image.

```
$ sudo docker pull training/docker-fundamentals-image
Pulling repository training/docker-fundamentals-image
8144a5b2bc0c: Download complete
511136ea3c5a: Download complete
8abc22fbb042: Download complete
58394af37342: Download complete
6ea7713376aa: Download complete
```

- Download the ubuntu image

```
$ sudo docker pull ubuntu:latest
Pulling repository ubuntu
9f676bd305a4: Download complete
9cd978db300e: Download complete
bac448df371d: Downloading
[============> ] 10.04 MB/39.93 MB 23s
e7d62a8128cf: Downloading
[======> ] 8.982 MB/68.32 MB
1m21s
f323cf34fd77: Download complete
```

Show Current Images

- Look at what images are on our host now.

```
$ docker images
REPOSITORY TAG IMAGE ID CREATED VIRTUAL SIZE
training/docker-fundamentals-image latest 8144a5b2bc0c 5 days ago
835 MB
ubuntu 13.10 9f676bd305a4 7 weeks ago 178 MB
ubuntu saucy 9f676bd305a4 7 weeks ago 178 MB
ubuntu raring eb601b8965b8 7 weeks ago 166.5 MB
ubuntu 13.04 eb601b8965b8 7 weeks ago 166.5 MB
ubuntu 12.10 5ac751e8d623 7 weeks ago 161 MB
ubuntu quantal 5ac751e8d623 7 weeks ago 161 MB
ubuntu 10.04 9cc9ea5ea540 7 weeks ago 180.8 MB
ubuntu lucid 9cc9ea5ea540 7 weeks ago 180.8 MB
ubuntu 12.04 9cd978db300e 7 weeks ago 204.4 MB
ubuntu latest 9cd978db300e 7 weeks ago 204.4 MB
ubuntu precise 9cd978db300e 7 weeks ago 204.4 MB
```

Types of Container

- Containers are created with the `docker run` command.
- Containers have two modes they run in:
  - Daemonized.
  - Interactive.
- Daemonized containers run in the background.
  - The `docker run` command is launched with the `-d` command line flag.
  - The container runs until it is stopped or killed.
- Interactive containers run in the foreground.
- Attached a pseudo-terminal, i.e. let you get input and output from the container.
- The container also runs until its controlling process stops or it is stopped or killed.

Launching an Interactive Container

- Create a new container from the ubuntu image:

```
$ docker run -i -t ubuntu /bin/bash
root@268e59b5754c:/#
```

- `-i` flag sets Docker's mode to interactive.
- `-t` flag creates a pseudo terminal (or PTY) in the container.
- We've specified the `ubuntu` image from which to create our container.
- We passed a command to run inside the container, `/bin/bash`.
- That command has launched a Bash shell inside our container.
- The hexadecimal number after `root@` is the container's identifier.(The actual ID is longer than that. Docker truncates it for convenience, just like `git` will show shorter ID instead of full hashes.)
- On the above prompt, we are inside our container.

Inside the Container

```
root@268e59b5754c:/#
```

- Let's run a command.

```
root@268e59b5754c:/# uname -rn
268e59b5754c 3.10.40-50.136.amzn1.x86_64
```

- Now let's exit the container.

```
root@268e59b5754c:/# exit
```

- After we run exit the container stops.
- Check the kernel version and hostname again, *outside* the container:

```
[docker@ip-172-31-47-238 ~]$ uname -rn
ip-172-31-47-238.ec2.internal 3.10.40-50.136.amzn1.x86_64
```

- The kernel version might be the same. Hostname is different.

Container Status

- You can see container status using the `docker ps` command. e.g.:

```
$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
```

- The `docker ps` command only shows running containers.
- Since the container has stopped we can show it by adding the `-l` flag. "l" for last. This shows the last run container, running or stopped.

```
$ docker ps -l
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
a2d4b003d7b6 ubuntu:12.04 /bin/bash 5 minutes ago Exit 0 sad_pare
```

- We can also use the `docker ps` command with the `-a` flag. The `-a` flag tells Docker to list all containers both running and stopped.

```
$ docker ps -a
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
a2d4b003d7b6 ubuntu:14.04 /bin/bash 5 minutes ago Exit 0 sad_pare
acc65c24dceb training/webapp:latest python -m SimpleHTTP 41 minutes 5000/tcp,
0.0.0.0:49154->8000/tcp furious_perlman
833daa3d9708 training/webapp:latest python -m SimpleHTTP 44 minutes ago
```

Information provide by `docker ps`

- A lot of data is returned by the `docker ps` command.

```
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
a2d4b003d7b6 ubuntu:14.04 /bin/bash 5 minutes ago Exit 0 sad_pare
```

- CONTAINER ID is a unique identifier generated by Docker for our container.
- You can use it to manage the container (e.g. stop it, examine it...)
- IMAGE is the image used to create that container.
- We did `docker run ubuntu`, and Docker selected ubuntu:14.04.
- COMMAND is the exact command that we asked Docker to run: `/bin/bash`.
- You can name your containers (with the `--name` option). If you don't, Docker will generate a random name for you, like `sad_pare`.
- That name shows up in the NAMES column.
- To get the ID of the last container, type

```
$ docker ps -l -q
ee9165307acc
```

`-l` means "show only the last container started".

`-q` means "show only the short ID of the container".

docker inspect command

- We can get a lot more information about our container by using the docker inspect command.

```
$ docker inspect $(docker ps -l -q) | less
[{
    "ID": "ee9165307accee9165307accee9165307acc",#  <yourContainerID>
    "Created": "2014-03-15T22:05:42.73203576Z",
    "Path": "/bin/bash",
    "Args": [],
    "Config": {
    "Hostname": "<yourContainerID>",
    "Domainname": "",
    "User": "",
    . . .
}]
```

- The full ID of the container is longer.
- We can also use the docker inspect command to find specific things about our container, for example:

```
$ docker inspect --format='{{.State.Running}}' $(docker ps -l -q)
false
```

`docker` shorthands

- We could use the ID of the container to get its properties

```
$ docker inspect <yourContainerID>
```

- Docker lets us type just the first characters of the ID.

```
$ docker inspect a2d4
```

Restarting a container

- You can (re-)start a stopped container using its ID.

```
$ docker start <yourContainerID>
<yourContainerID>
```

- Or using its name.

```
$ docker start sad_pare
sad_pare
```

- The container will be restarted using the same options you launched it with.

Attaching to a running container

- Once the container is started you can attach to it. In our case this will attach us to the Bash shell we launched when we ran the container initially.

```
$ docker attach <yourContainerID>
root@<yourContainerID>:/#
```

- *Note:* if the prompt is not displayed after running `docker attach`, just press "`Enter`" one more time. The prompt should then appear.
- You can also attach to the container using its name.

```
$ docker attach sad_pare
root@<yourContainerID>:/#
```

- If we ran `exit` here the container would stop again because the `/bin/bash` process would be ended.
- You can detach from the running container using `<CTRL+p><CTRL+q>`.
- There's also a shortcut we can use that combines the docker start and docker attach

```
$ docker start -a <yourContainerID>
```

- The `-a` flag combines the function of `docker attach` when running the `docker start`