**Introduction**

Planning, deploying, and running Azure Databricks (ADB) at scale requires one to make many architectural decisions.

While each ADB deployment is unique to an organization's needs we have found that some patterns are common across most successful ADB projects. Unsurprisingly, these patterns are also in-line with modern Cloud-centric development best practices.

This short guide summarizes these patterns into prescriptive and actionable best practices for Azure Databricks. We follow a logical path of planning the infrastructure, provisioning the workspaces,developing Azure Databricks applications, and finally, running Azure Databricks in production.

The audience of this guide are system architects, field engineers, and development teams of customers, Microsoft, and Databricks. Since the Azure Databricks product goes through fast iteration cycles, we have avoided recommendations based on roadmap or Private Preview features.

Our recommendations should apply to a typical Fortune 500 enterprise with at least intermediate level of Azure and Databricks knowledge. We've also classified each recommendation according to its likely impact on solution's quality attributes. Using the **Impact** factor, you can weigh the recommendation against other competing choices. Example: if the impact is classified as “Very High”, the implications of not adopting the best practice can have a significant impact on your deployment.

**Important Note**: This guide is intended to be used with the detailed [Azure Databricks Documentation](https://docs.azuredatabricks.net/index.html)

**Scalable ADB Deployments: Guidelines for Networking, Security, and Capacity Planning**

Azure Databricks (ADB) deployments for very small organizations, PoC applications, or for personal education hardly require any planning. You can spin up a Workspace using Azure Portal in a matter of minutes, create a Notebook, and start writing code.

Enterprise-grade large scale deployments are a different story altogether. Some upfront planning is necessary to manage Azure Databricks deployments across large teams. In particular, you need to understand:

* Networking requirements of Databricks
* The number and the type of Azure networking resources required to launch clusters
* Relationship between Azure and Databricks jargon: Subscription, VNet., Workspaces, Clusters, Subnets, etc.
* Overall Capacity Planning process: where to begin, what to consider?

Let’s start with a short Azure Databricks 101 and then discuss some best practices for scalable and secure deployments.

**Azure Databricks 101**

ADB is a Big Data analytics service. Being a Cloud Optimized managed [PaaS](https://azure.microsoft.com/en-us/overview/what-is-paas/) offering, it is designed to hide the underlying distributed systems and networking complexity as much as possible from the end user. It is backed by a team of support staff who monitor its health, debug tickets filed via Azure, etc. This allows ADB users to focus on developing value generating apps rather than stressing over infrastructure management.

You can deploy ADB using Azure Portal or using [ARM templates](https://docs.microsoft.com/en-us/azure/azure-resource-manager/resource-group-overview#template-deployment). One successful ADB deployment produces exactly one Workspace, a space where users can log in and author analytics apps. It comprises the file browser, notebooks, tables, clusters, [DBFS](https://docs.azuredatabricks.net/user-guide/dbfs-databricks-file-system.html#dbfs) storage, etc. More importantly, Workspace is a fundamental isolation unit in Databricks. All workspaces are completely isolated from each other.

Each workspace is identified by a globally unique 53-bit number, called ***Workspace ID or Organization ID***. The URL that a customer sees after logging in always uniquely identifies the workspace they are using:

[*https://regionName.azuredatabricks.net/?o=workspaceId*](https://regionname.azuredatabricks.net/?o=workspaceId)

Example: [*https://eastus2.azuredatabricks.net/?o=12345*](https://eastus2.azuredatabricks.net/?o=12345)

Azure Databricks uses [Azure Active Directory (AAD)](https://docs.microsoft.com/en-us/azure/active-directory/fundamentals/active-directory-whatis) as the exclusive Identity Provider and there’s a seamless out of the box integration between them. This makes ADB tightly integrated with Azure just like its other core services. Any AAD member assigned to the Owner or Contributor role can deploy Databricks and is automatically added to the ADB members list upon first login. If a user is not a member or guest of the Active Directory tenant, they can’t login to the workspace. Granting access to a user in another tenant (for example, if contoso.com wants to collaborate with adventure-works.com users) does work because those external users are added as guests to the tenant hosting Azure Databricks.

Azure Databricks comes with its own user management interface. You can create users and groups in a workspace, assign them certain privileges, etc. While users in AAD are equivalent to Databricks users, by default AAD roles have no relationship with groups created inside ADB, unless you use [SCIM](https://docs.azuredatabricks.net/administration-guide/admin-settings/scim/aad.html) for provisioning users and groups. With SCIM, you can import both groups and users from AAD into Azure Databricks, and the synchronization is automatic after the initial import. ADB also has a special group called ***Admins***, not to be confused with AAD’s role Admin.

The first user to login and initialize the workspace is the workspace ***owner***, and they are automatically assigned to the Databricks admin group. This person can invite other users to the workspace, add them as admins, create groups, etc. The ADB logged in user’s identity is provided by AAD, and shows up under the user menu in Workspace:

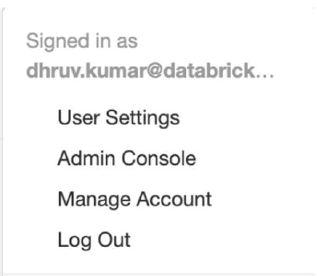
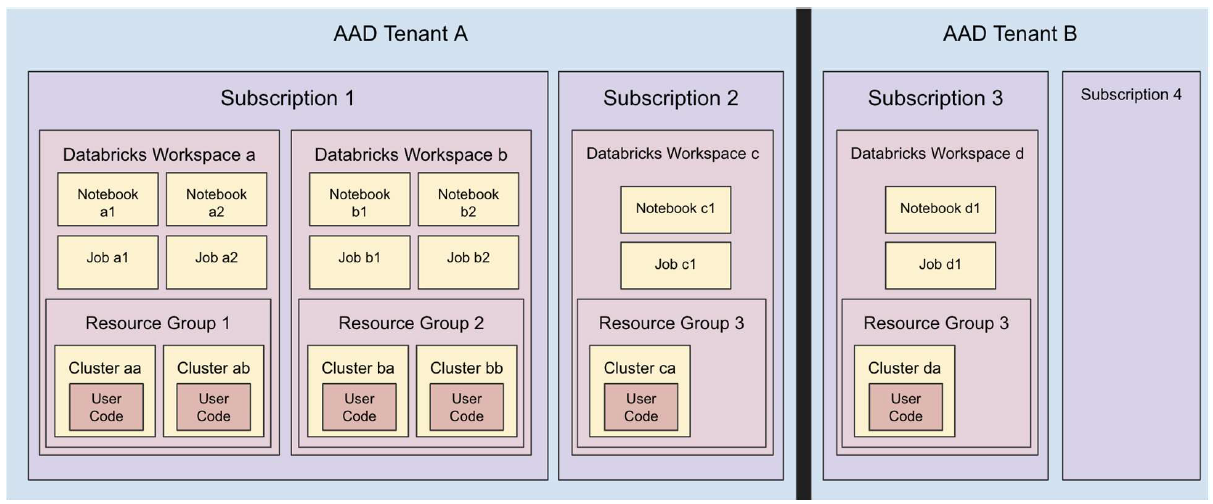
[](https://github.com/Azure/AzureDatabricksBestPractices/blob/master/Figure1.PNG)

Figure 1: Databricks user menu

Multiple clusters can exist within a workspace, and there’s a one-to-many mapping between a Subscription to Workspaces, and further, from one Workspace to multiple Clusters.

[](https://github.com/Azure/AzureDatabricksBestPractices/blob/master/Figure3.PNG)

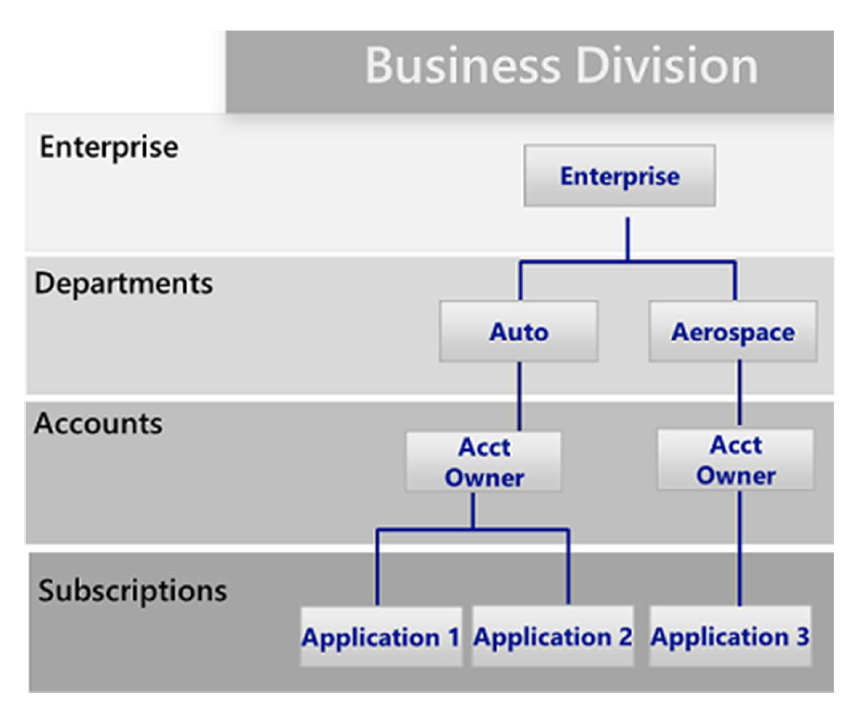
\*Figure 2: Relationship Between AAD, Workspace, Resource Groups, and Clusters

With this basic understanding let’s discuss how to plan a typical ADB deployment. We first grapple with the issue of how to divide workspaces and assign them to users and teams.

**Map Workspaces to Business Divisions**

*Impact: Very High*

How many workspaces do you need to deploy? The answer to this question depends a lot on your organization’s structure. We recommend that you assign workspaces based on a related group of people working together collaboratively. This also helps in streamlining your access control matrix within your workspace (folders, notebooks etc.) and also across all your resources that the workspace interacts with (storage, related data stores like Azure SQL DB, Azure SQL DW etc.). This type of division scheme is also known as the [Business Unit Subscription](https://docs.microsoft.com/en-us/azure/cloud-adoption-framework/decision-guides/subscriptions/) design pattern and it aligns well with the Databricks chargeback model.

[](https://github.com/Azure/AzureDatabricksBestPractices/blob/master/Figure2.PNG)

*Figure 3: Business Unit Subscription Design Pattern*

**Deploy Workspaces in Multiple Subscriptions to Honor Azure Capacity Limits**

*Impact: Very High*

Customers commonly partition workspaces based on teams or departments and arrive at that division naturally. But it is also important to partition keeping Azure Subscription and ADB Workspace limits in mind.

**Databricks Workspace Limits**

Azure Databricks is a multitenant service and to provide fair resource sharing to all regional customers, it imposes limits on API calls. These limits are expressed at the Workspace level and are due to internal ADB components. For instance, you can only run up to 150 concurrent jobs in a workspace. Beyond that, ADB will deny your job submissions. There are also other limits such as max hourly job submissions, max notebooks, etc.

Key workspace limits are:

* The maximum number of jobs that a workspace can create in an hour is **1000**
* At any time, you cannot have more than **150 jobs** simultaneously running in a workspace
* There can be a maximum of **150 notebooks or execution contexts** attached to a cluster
* The maximum number secret scopes per workspace is 100

**Azure Subscription Limits**

Next, there are [Azure limits](https://docs.microsoft.com/en-us/azure/azure-subscription-service-limits) to consider since ADB deployments are built on top of the Azure infrastructure.

Key Azure limits are:

* Storage accounts per region per subscription: **250**
* Maximum egress for general-purpose v2 and Blob storage accounts (all regions): **50 Gbps**
* VMs per subscription per region: **25,000**
* Resource groups per subscription: **980**

These limits are at this point in time and might change going forward. Some of them can also be increased if needed. For more help in understanding the impact of these limits or options of increasing them, please contact Microsoft or Databricks technical architects.

***Due to scalability reasons, we highly recommend separating the production and dev/stage environments into separate subscriptions.***

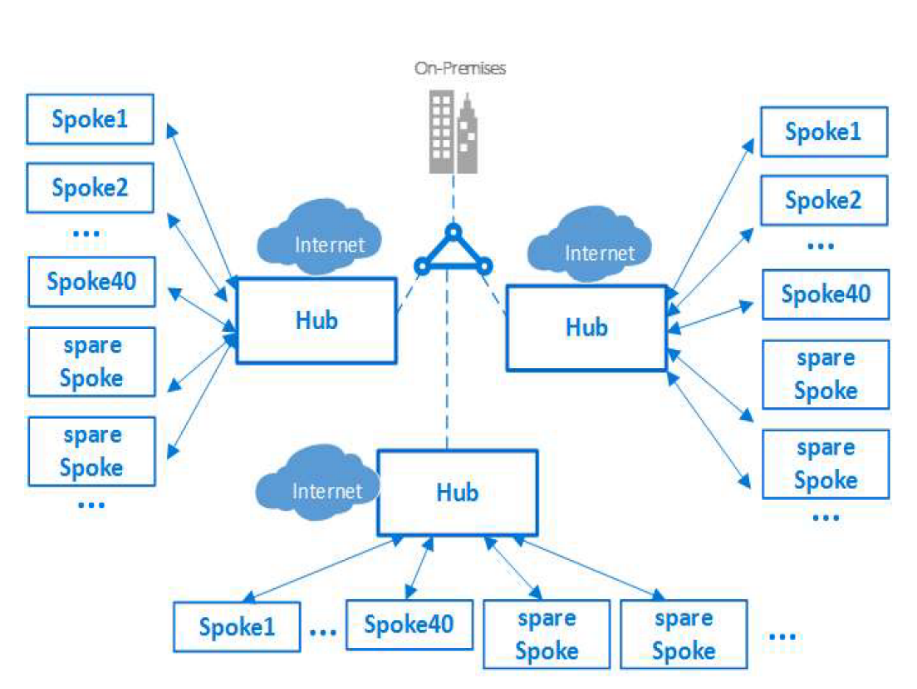
**Consider Isolating Each Workspace in its own VNet**

*Impact: Low*

While you can deploy more than one Workspace in a VNet by keeping the associated subnet pairs separate from other workspaces, we recommend that you should only deploy one workspace in any Vnet. Doing this perfectly aligns with the ADB's Workspace level isolation model. Most often organizations consider putting multiple workspaces in the same Vnet so that they all can share some common networking resource, like DNS, also placed in the same Vnet because the private address space in a vnet is shared by all resources. You can easily achieve the same while keeping the Workspaces separate by following the [hub and spoke model](https://docs.microsoft.com/en-us/azure/architecture/reference-architectures/hybrid-networking/hub-spoke) and using Vnet Peering to extend the private IP space of the workspace Vnet. Here are the steps:

1. Deploy each Workspace in its own spoke VNet.
2. Put all the common networking resources in a central hub Vnet, such as your custom DNS server.
3. Join the Workspace spokes with the central networking hub using [Vnet Peering](https://docs.azuredatabricks.net/administration-guide/cloud-configurations/azure/vnet-peering.html)

More information: [Azure Virtual Datacenter: a network perspective](https://docs.microsoft.com/en-us/azure/architecture/vdc/networking-virtual-datacenter#topology)

[](https://github.com/Azure/AzureDatabricksBestPractices/blob/master/Figure4.PNG)

*Figure 4: Hub and Spoke Model*

**Select the Largest Vnet CIDR**

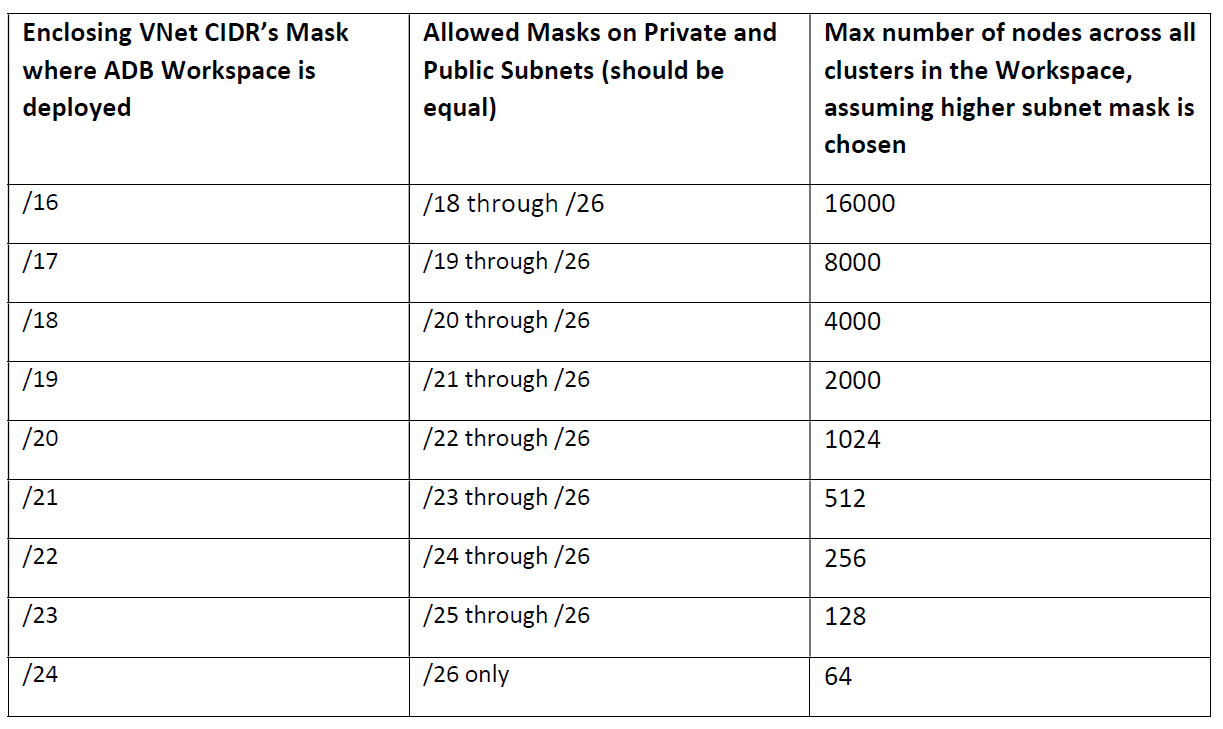
*Impact: Very High*

***This recommendation only applies if you're using the Bring Your Own Vnet feature.***

Recall the each Workspace can have multiple clusters. The total capacity of clusters in each workspace is a function of the masks used for the workspace's enclosing Vnet and the pair of subnets associated with each cluster in the workspace. The masks can be changed if you use the [Bring Your Own Vnet](https://docs.azuredatabricks.net/administration-guide/cloud-configurations/azure/vnet-inject.html#vnet-inject) feature as it gives you more control over the networking layout. It is important to understand this relationship for accurate capacity planning.

* Each cluster node requires 1 Public IP and 2 Private IPs
* These IPs are logically grouped into 2 subnets named “public” and “private”
* For a desired cluster size of X: number of Public IPs = X, number of Private IPs = 4X
* The 4X requirement for Private IPs is due to the fact that for each deployment:
  + Half of address space is reserved for future use
  + The other half is equally divided into the two subnets: private and public
* The size of private and public subnets thur determines total number of VMs available for clusters
  + /22 mask is larger than /23, so setting private and public to /22 will have more VMs available for creating clusters, than say /23 or below
* But, because of the address space allocation scheme, the size of private and public subnets is constrained by the VNet’s CIDR
* The allowed values for the enclosing VNet CIDR are from /16 through /24
* The private and public subnet masks must be:
  + Equal
  + At least two steps down from enclosing VNet CIDR mask
  + Must be greater than /26

With this info, we can quickly arrive at the table below, showing how many nodes one can use across all clusters for a given VNet CIDR. It is clear that selection of VNet CIDR has far reaching implications in terms of maximum cluster size.

[](https://github.com/Azure/AzureDatabricksBestPractices/blob/master/Table1.PNG)

**Azure Databricks Deployment with limited private IP addresses**

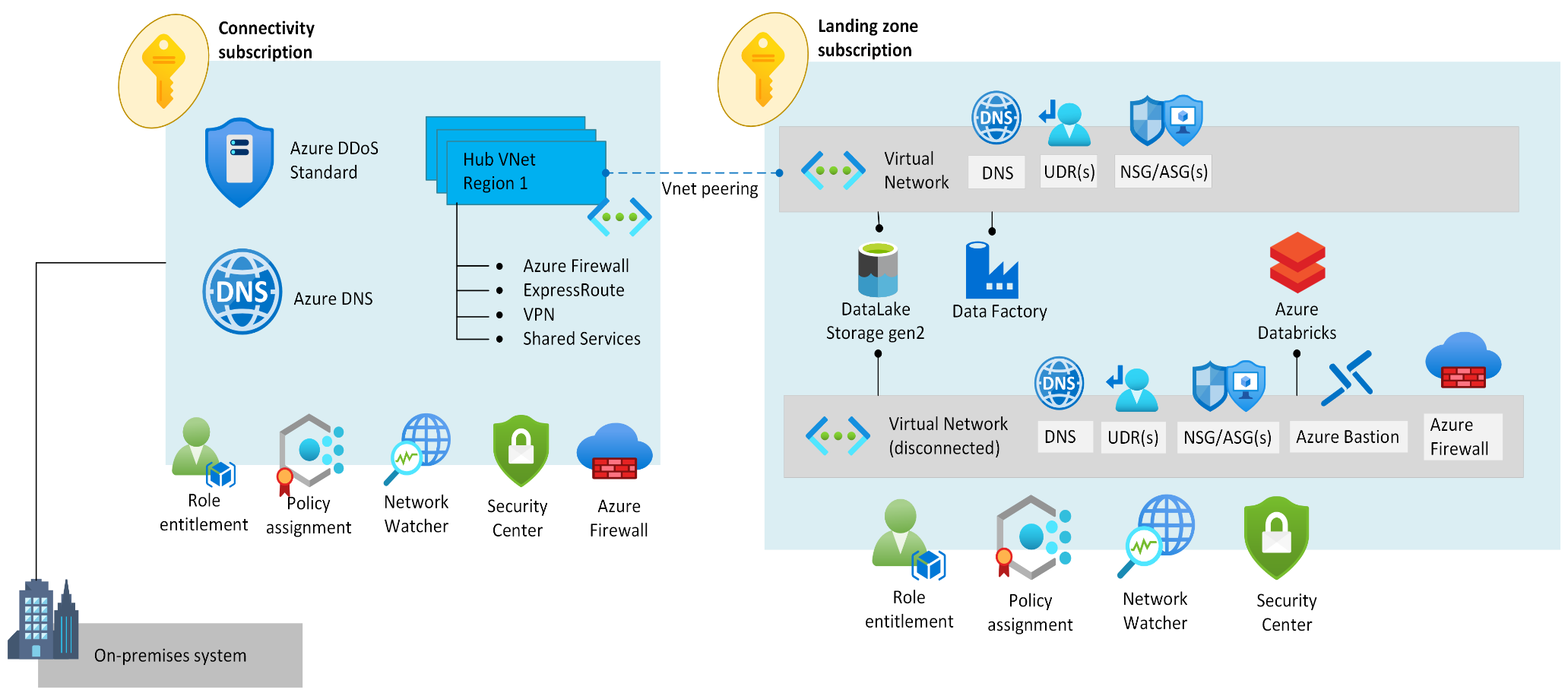
*Impact: High*

Depending where data sources are located, Azure Databricks can be deployed in a connected or disconnected scenario. In a connected scenario, Azure Databricks must be able to reach directly data sources located in Azure VNets or on-premises locations. In a disconnected scenario, data can be copied to a storage platform (such as an Azure Data Lake Storage account), to which Azure Databricks can be connected to using mount points. ***This section will cover a scenario to deploy Azure Databricks when there are limited private IP addresses and Azure Databricks can be configured to access data using mount points (disconnected scenario).***

Many multi-national enterprise organizations are building platforms in Azure, based on the hub and spoke network architecture, which is a model that maps to the recommended Azure Databricks deployments, which is to deploy only one workspace in any VNet by implementing the hub and spoke network architecture. Workspaces are deployed on the spokes, while shared networking and security resources such as ExpressRoute connectivity or DNS infrastructure is deployed in the hub. Customer who have exhausted (or are near to exhaust) RFC1918 IP address ranges, have to optimize address space for spoke VNets, and may only be able to provide small VNets for most cases (/25 or smaller), and only in exceptional cases they may provide a larger VNet (such as a /24).

As the smallest Azure Databricks deployment requires a /24 VNet, such customers require an alternative solution, so that the business can deploy one or multiple Azure Databricks clusters across multiple VNets (as required by the business), but also, they should be able to create larger clusters, which would require larger VNet address space.

A recommended Azure Databricks implementation, which would ensure minimal RFC1918 addresses are used, while at the same time, would allow the business users to deploy as many Azure Databricks clusters as they want and as small or large as they need them, consist on the following environments within the same Azure subscription as depicted in the picture below:

[](https://github.com/Azure/AzureDatabricksBestPractices/blob/master/Figure8.png)

*Figure 8: Network Topology*

As the diagram depicts, the business application subscription where Azure Databricks will be deployed, has two VNets, one that is routable to on-premises and the rest of the Azure environment (this can be a small VNet such as /26), and includes the following Azure data resources: Azure Data Factory and ADLS Gen2 (via Private Endpoint).

***Note: While we use Azure Data Factory on this implementation, any other service that can perform similar functionality could be used.***

The other VNet is fully disconnected and is not routable to the rest of the environment, and on this VNet Databricks and optionally Azure Bastion (to be able to perform management via jumpboxes) is deployed, as well as a Private Endpoint to the ADLS Gen2 storage, so that Databricks can retrieve data for ingestion. This setup is described in further details below:

**Connected (routable environment)**

* In a business application subscription, deploy a VNet with RFC1918 addresses which is fully routable in Azure and cross-premises via ExpressRoute. This VNet can be a small VNet, such as /26 or /27.
* This VNet, is connected to a central hub VNet via VNet peering to have connectivity across Azure and on-premises via ExpressRoute or VPN.
* UDR with default route (0.0.0.0/0) points to a central NVA (for example, Azure Firewall) for internet outbound traffic.
* NSGs are configured to block inbound traffic from the internet.
* Azure Data Lake (ADLS) Gen2 is deployed in the business application subscription.
* A Private Endpoint is created on the VNet to make ADLS Gen 2 storage accessible from on-premises and from Azure VNets via a private IP address.
* Azure Data Factory will be responsible for the process of moving data from the source locations (other spoke VNets or on-premises) into the ADLS Gen2 store (accessible via Private Endpoint).
* Azure Data Factory (ADF) is deployed on this routable VNet
  + Azure Data Factory components require a compute infrastructure to run on and this is referred to as Integration Runtime. In the mentioned scenario, moving data from on-premises data sources to Azure Data Services (accessible via Private Endpoint), it is required a Self-Hosted Integration Runtime.
  + The Self-Hosted Integration Runtime needs to be installed on an Azure Virtual Machine inside the routable VNET in order to allow Azure Data Factory to communicate with the source data and destination data.
  + Considering this, Azure Data Factory only requires 1 IP address (and maximum up to 4 IP addresses) in the VNet (via the integration runtime).

**Disconnected (non-routable environment)**

* In the same business application subscription, deploy a VNet with any RFC1918 address space that is desired by the application team (for example, 10.0.0.0/16)
* This VNet is not going to be connected to the rest of the environment. In other words, this will be a disconnected and fully isolated VNet.
* This VNet includes 3 required and 3 optional subnets:
  + 2x of them dedicated exclusively to the Azure Databricks Workspace (private-subnet and public-subnet)
  + 1x which will be used for the private link to the ADLS Gen2
  + (Optional) 1x for Azure Bastion
  + (Optional) 1x for jumpboxes
  + (Optional but recommended) 1x for Azure Firewall (or other network security NVA).
* Azure Databricks is deployed on this disconnected VNet.
* Azure Bastion is deployed on this disconnected VNet, to allow Azure Databricks administration via jumpboxes.
* Azure Firewall (or another network security NVA) is deployed on this disconnected VNet to secure internet outbound traffic.
* NSGs are used to lockdown traffic across subnets.
* 2x Private Endpoints are created on this disconnected VNet to make the ADLS Gen2 storage accessible for the Databricks cluster:
  + 1x private endpoint having the target sub-resource *blob*
  + 1x private endpoint having the target sub-resource *dfs*
* Databricks integrates with ADLS Gen2 storage for data ingestion

**Do not Store any Production Data in Default DBFS Folders**

*Impact: High*

This recommendation is driven by security and data availability concerns. Every Workspace comes with a default DBFS, primarily designed to store libraries and other system-level configuration artifacts such as Init scripts. You should not store any production data in it, because:

1. The lifecycle of default DBFS is tied to the Workspace. Deleting the workspace will also delete the default DBFS and permanently remove its contents.
2. One can't restrict access to this default folder and its contents.

***This recommendation doesn't apply to Blob or ADLS folders explicitly mounted as DBFS by the end user***

**More Information:** [Databricks File System](https://docs.azuredatabricks.net/user-guide/dbfs-databricks-file-system.html)

**Always Hide Secrets in a Key Vault**

*Impact: High*

It is a significant security risk to expose sensitive data such as access credentials openly in Notebooks or other places such as job configs, init scripts, etc. You should always use a vault to securely store and access them. You can either use ADB’s internal Key Vault for this purpose or use Azure’s Key Vault (AKV) service.

If using Azure Key Vault, create separate AKV-backed secret scopes and corresponding AKVs to store credentials pertaining to different data stores. This will help prevent users from accessing credentials that they might not have access to. Since access controls are applicable to the entire secret scope, users with access to the scope will see all secrets for the AKV associated with that scope.

**More Information:**

[Create an Azure Key Vault-backed secret scope](https://docs.azuredatabricks.net/user-guide/secrets/secret-scopes.html)

[Example of using secret in a notebook](https://docs.azuredatabricks.net/user-guide/secrets/example-secret-workflow.html)

[Best practices for creating secret scopes](https://docs.azuredatabricks.net/user-guide/secrets/secret-acl.html)

**Deploying Applications on ADB: Guidelines for Selecting, Sizing, and Optimizing Clusters Performance**

***"Any organization that designs a system will inevitably produce a design whose structure is a copy of the organization's communication structure."*** Mead Conway

After understanding how to provision the workspaces, best practices in networking, etc., let’s put on the developer’s hat and see the design choices typically faced by them:

* What type of clusters should I use?
* How many drivers and how many workers?
* Which Azure VMs should I select?

In this chapter we will address such concerns and provide our recommendations, while also explaining the internals of Databricks clusters and associated topics. Some of these ideas seem counterintuitive but they will all make sense if you keep these important design attributes of the ADB service in mind:

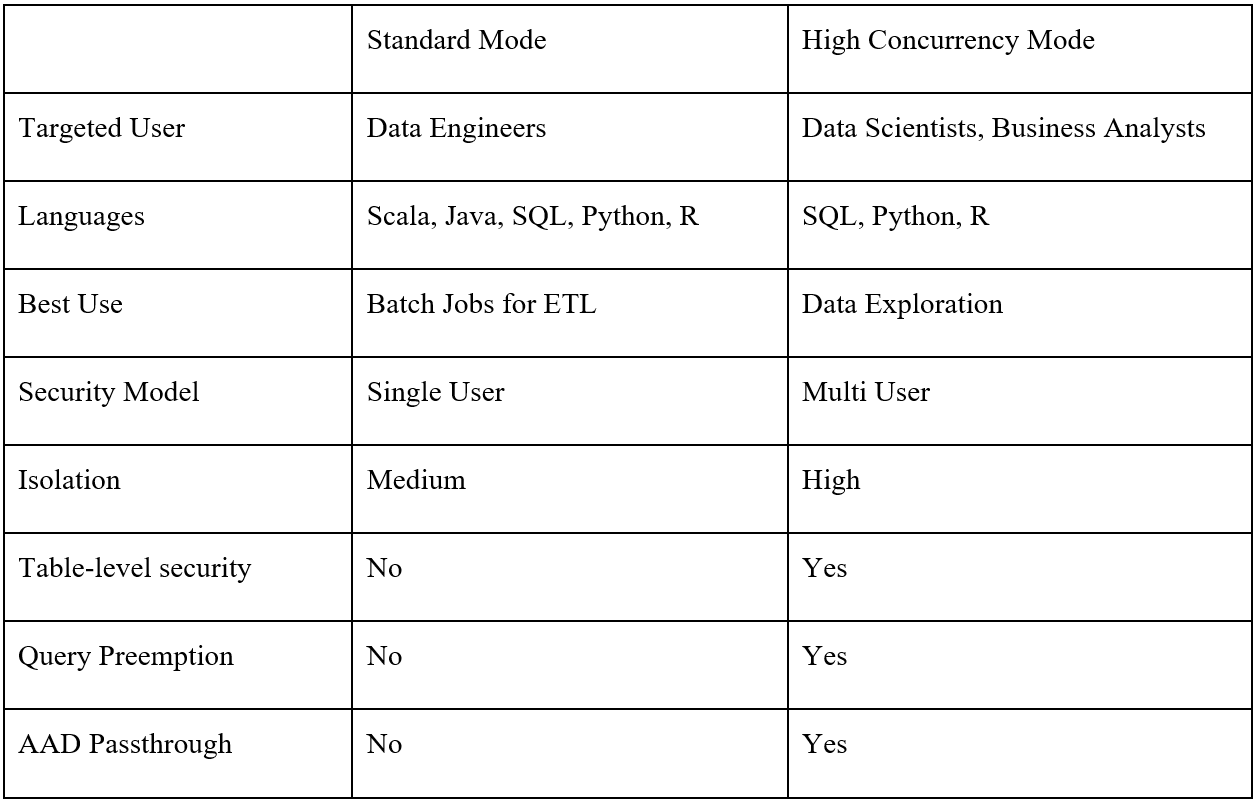
1. **Cloud Optimized:** Azure Databricks is a product built exclusively for cloud environments, like Azure. No on-prem deployments currently exist. It assumes certain features are provided by the Cloud, is designed keeping Cloud best practices, and conversely, provides Cloud-friendly features.
2. **Platform/Software as a Service Abstraction:** ADB sits somewhere between the PaaS and SaaS ends of the spectrum, depending on how you use it. In either case ADB is designed to hide infrastructure details as much as possible so the user can focus on application development. It is not, for example, an IaaS offering exposing the guts of the OS Kernel to you.
3. **Managed Service:** ADB guarantees a 99.95% uptime SLA. There’s a large team of dedicated staff members who monitor various aspects of its health and get alerted when something goes wrong. It is run like an always-on website and Microsoft and Databricks system operations team strives to minimize any downtime.

These three attributes make ADB very different than other Spark platforms such as HDP, CDH, Mesos, etc. which are designed for on-prem datacenters and allow the user complete control over the hardware. The concept of a cluster is therefore pretty unique in Azure Databricks. Unlike YARN or Mesos clusters which are just a collection of worker machines waiting for an application to be scheduled on them, clusters in ADB come with a pre-configured Spark application. ADB submits all subsequent user requests like notebook commands, SQL queries, Java jar jobs, etc. to this primordial app for execution.

Under the covers Databricks clusters use the lightweight Spark Standalone resource allocator.

When it comes to taxonomy, ADB clusters are divided along the notions of “type”, and “mode.” There are two ***types*** of ADB clusters, according to how they are created. Clusters created using UI and [Clusters API](https://docs.azuredatabricks.net/api/latest/clusters.html) are called Interactive Clusters, whereas those created using [Jobs API](https://docs.azuredatabricks.net/api/latest/jobs.html) are called Jobs Clusters. Further, each cluster can be of two ***modes***: Standard and High Concurrency. Regardless of types or mode, all clusters in Azure Databricks can automatically scale to match the workload, using a feature known as [Autoscaling](https://docs.azuredatabricks.net/user-guide/clusters/sizing.html#cluster-size-and-autoscaling).

*Table 2: Cluster modes and their characteristics*

[](https://github.com/Azure/AzureDatabricksBestPractices/blob/master/Table2.PNG)

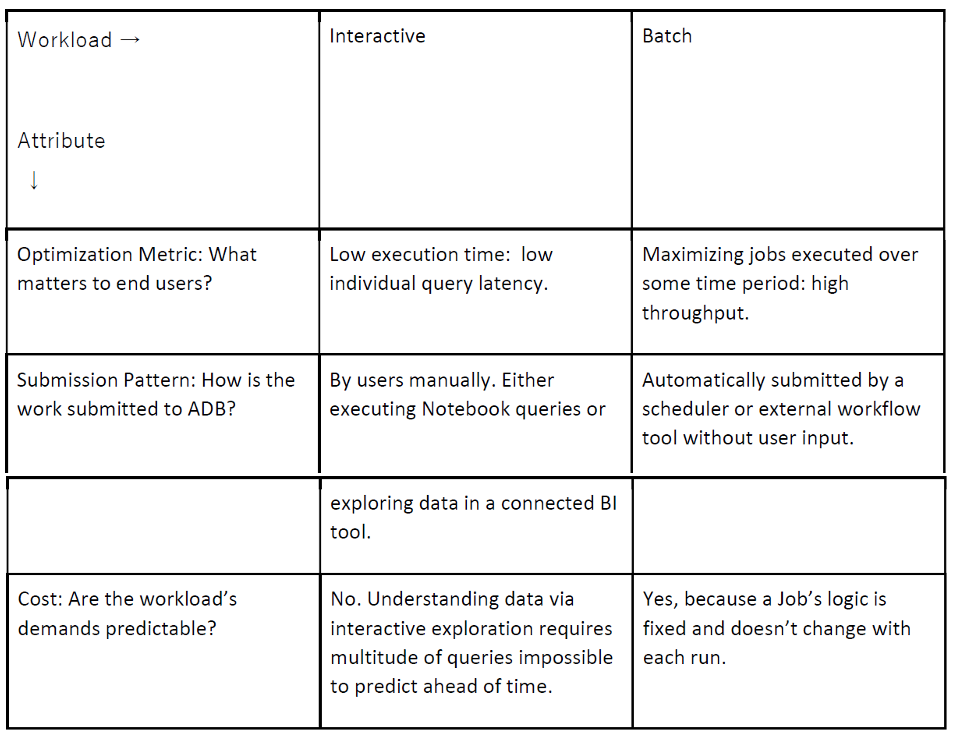
**Support Interactive Analytics Using Shared High Concurrency Clusters**

*Impact: Medium*

There are three steps for supporting Interactive workloads on ADB:

1. Deploy a shared cluster instead of letting each user create their own cluster.
2. Create the shared cluster in High Concurrency mode instead of Standard mode.
3. Configure security on the shared High Concurrency cluster, using **one** of the following options:
   * Turn on [AAD Credential Passthrough](https://docs.azuredatabricks.net/administration-guide/cloud-configurations/azure/credential-passthrough.html#enabling-azure-ad-credential-passthrough-to-adls) if you’re using ADLS
   * Turn on Table Access Control for all other stores

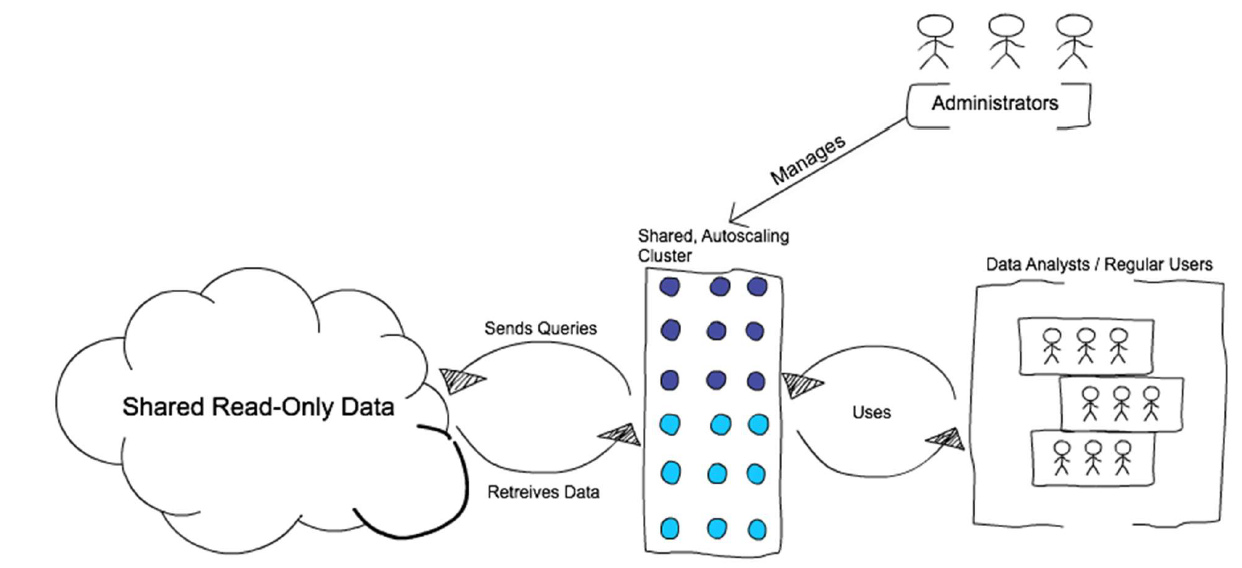
To understand why, let’s quickly see how interactive workloads are different from batch workloads:

*Table 3: Batch vs. Interactive workloads* [](https://github.com/Azure/AzureDatabricksBestPractices/blob/master/Table3.PNG)

Because of these differences, supporting Interactive workloads entails minimizing cost variability and optimizing for latency over throughput, while providing a secure environment. These goals are satisfied by shared High Concurrency clusters with Table access controls or AAD Passthrough turned on (in case of ADLS):

1. **Minimizing Cost:** By forcing users to share an autoscaling cluster you have configured with maximum node count, rather than say, asking them to create a new one for their use each time they log in, you can control the total cost easily. The max cost of shared cluster can be calculated by assuming it is running X hours at maximum size with the particular VMs. It is difficult to achieve this if each user is given free reign over creating clusters of arbitrary size and VMs.
2. **Optimizing for Latency:** Only High Concurrency clusters have features which allow queries from different users share cluster resources in a fair, secure manner. HC clusters come with Query Watchdog, a process which keeps disruptive queries in check by automatically pre-empting rogue queries, limiting the maximum size of output rows returned, etc.
3. **Security:** Table Access control feature is only available in High Concurrency mode and needs to be turned on so that users can limit access to their database objects (tables, views, functions, etc.) created on the shared cluster. In case of ADLS, we recommend restricting access using the AAD Credential Passthrough feature instead of Table Access Controls.

***If you’re using ADLS, we recommend AAD Credential Passthrough instead of Table Access Control for easy manageability.***

[](https://github.com/Azure/AzureDatabricksBestPractices/blob/master/Figure5.PNG)

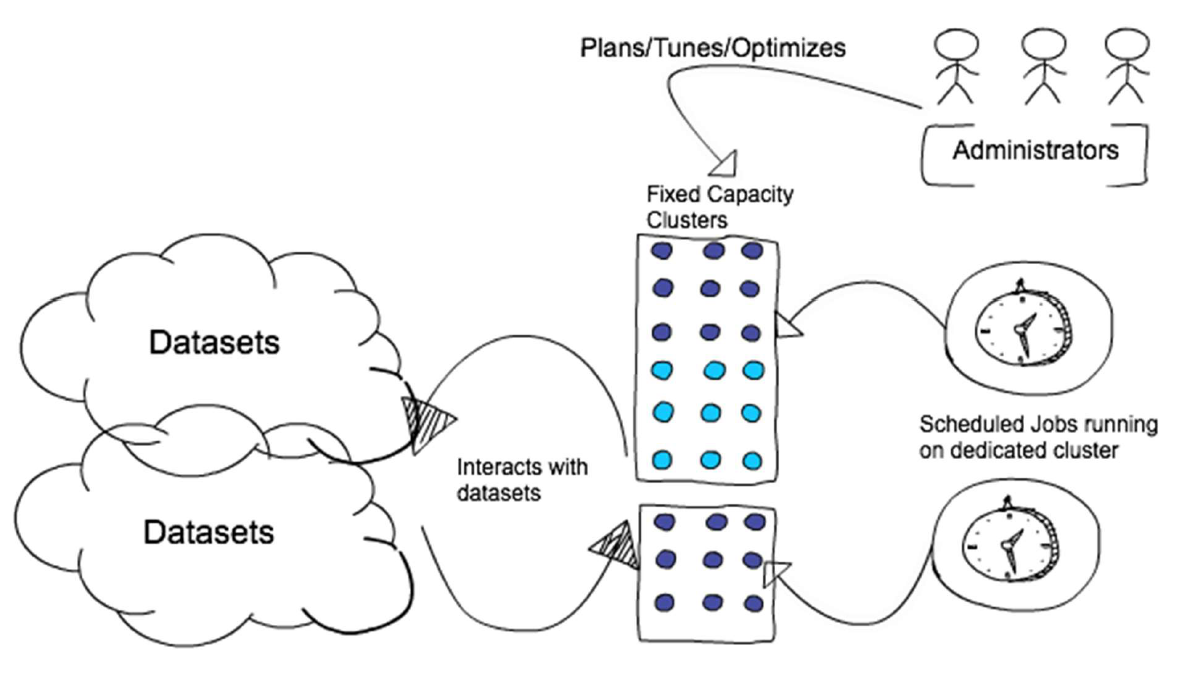
*Figure 5: Interactive clusters*

**Support Batch ETL Workloads With Single User Ephemeral Standard Clusters**

*Impact: Medium*

Unlike Interactive workloads, logic in batch Jobs is well defined and their cluster resource requirements are known *a priori*. Hence to minimize cost, there’s no reason to follow the shared cluster model and we recommend letting each job create a separate cluster for its execution. Thus, instead of submitting batch ETL jobs to a cluster already created from ADB’s UI, submit them using the Jobs APIs. These APIs automatically create new clusters to run Jobs and also terminate them after running it. We call this the **Ephemeral Job Cluster** pattern for running jobs because the clusters short life is tied to the job lifecycle.

Azure Data Factory uses this pattern as well - each job ends up creating a separate cluster since the underlying call is made using the [Runs-Submit Jobs API](https://docs.azuredatabricks.net/api/latest/jobs.html#runs-submit).

[](https://github.com/Azure/AzureDatabricksBestPractices/blob/master/Figure6.PNG)

*Figure 6: Ephemeral Job cluster*

Just like the previous recommendation, this pattern will achieve general goals of minimizing cost, improving the target metric (throughput), and enhancing security by:

1. **Enhanced Security:** ephemeral clusters run only one job at a time, so each executor’s JVM runs code from only one user. This makes ephemeral clusters more secure than shared clusters for Java and Scala code.
2. **Lower Cost:** if you run jobs on a cluster created from ADB’s UI, you will be charged at the higher Interactive DBU rate. The lower Data Engineering DBUs are only available when the lifecycle of job and cluster are same. This is only achievable using the Jobs APIs to launch jobs on ephemeral clusters.
3. **Better Throughput:** cluster’s resources are dedicated to one job only, making the job finish faster than while running in a shared environment.

For very short duration jobs (< 10 min) the cluster launch time (~ 7 min) adds a significant overhead to total execution time. Historically this forced users to run short jobs on existing clusters created by UI -- a costlier and less secure alternative. To fix this, ADB is coming out with a new feature called Instance Pools in Q3 2019 bringing down cluster launch time to 30 seconds or less.

**Favor Cluster Scoped Init Scripts over Global and Named scripts**

*Impact: High*

[Init Scripts](https://docs.azuredatabricks.net/user-guide/clusters/init-scripts.html) provide a way to configure cluster’s nodes and can be used in the following modes:

1. **Global:** by placing the init script in /databricks/init folder, you force the script’s execution every time any cluster is created or restarted by users of the workspace.
2. **Cluster Named (deprecated):** you can limit the init script to run only on for a specific cluster’s creation and restarts by placing it in /databricks/init/<cluster\_name> folder.
3. **Cluster Scoped:** in this mode the init script is not tied to any cluster by its name and its automatic execution is not a virtue of its dbfs location. Rather, you specify the script in cluster’s configuration by either writing it directly in the cluster configuration UI or storing it on DBFS and specifying the path in [Cluster Create API](https://docs.azuredatabricks.net/user-guide/clusters/init-scripts.html#cluster-scoped-init-script). Any location under DBFS /databricks folder except /databricks/init can be used for this purpose, such as: /databricks/<my-directory>/set-env-var.sh

You should treat Init scripts with *extreme* caution because they can easily lead to intractable cluster launch failures. If you really need them, please use the Cluster Scoped execution mode as much as possible because:

1. ADB executes the script’s body in each cluster node. Thus, a successful cluster launch and subsequent operation is predicated on all nodal init scripts executing in a timely manner without any errors and reporting a zero exit code. This process is highly error prone, especially for scripts downloading artifacts from an external service over unreliable and/or misconfigured networks.
2. Because Global and Cluster Named init scripts execute automatically due to their placement in a special DBFS location, it is easy to overlook that they could be causing a cluster to not launch. By specifying the Init script in the Configuration, there’s a higher chance that you’ll consider them while debugging launch failures.

**Use Cluster Log Delivery Feature to Manage Logs**

*Impact: Medium*

By default, Cluster logs are sent to default DBFS but you should consider sending the logs to a blob store location under your control using the [Cluster Log Delivery](https://docs.azuredatabricks.net/user-guide/clusters/log-delivery.html#cluster-log-delivery) feature. The Cluster Logs contain logs emitted by user code, as well as Spark framework’s Driver and Executor logs. Sending them to a blob store controlled by yourself is recommended over default DBFS location because:

1. ADB’s automatic 30-day default DBFS log purging policy might be too short for certain compliance scenarios. A blob store loction in your subscription will be free from such policies.
2. You can ship logs to other tools only if they are present in your storage account and a resource group governed by you. The root DBFS, although present in your subscription, is launched inside a Microsoft Azure managed resource group and is protected by a read lock. Because of this lock the logs are only accessible by privileged Azure Databricks framework code. However, constructing a pipeline to ship the logs to downstream log analytics tools requires logs to be in a lock-free location first.

**Choose Cluster VMs to Match Workload Class**

*Impact: High*

To allocate the right amount and type of cluster resource for a job, we need to understand how different types of jobs demand different types of cluster resources.

* **Machine Learning** - To train machine learning models it’s usually required cache all of the data in memory. Consider using memory optimized VMs so that the cluster can take advantage of the RAM cache. You can also use storage optimized instances for very large datasets. To size the cluster, take a % of the data set → cache it → see how much memory it used → extrapolate that to the rest of the data.
* **Streaming** - You need to make sure that the processing rate is just above the input rate at peak times of the day. Depending peak input rate times, consider compute optimized VMs for the cluster to make sure processing rate is higher than your input rate.
* **ETL** - In this case, data size and deciding how fast a job needs to be will be a leading indicator. Spark doesn’t always require data to be loaded into memory in order to execute transformations, but you’ll at the very least need to see how large the task sizes are on shuffles and compare that to the task throughput you’d like. To analyze the performance of these jobs start with basics and check if the job is by CPU, network, or local I/O, and go from there. Consider using a general purpose VM for these jobs.
* **Interactive / Development Workloads** - The ability for a cluster to auto scale is most important for these types of jobs. In this case taking advantage of the [Autoscaling feature](https://docs.azuredatabricks.net/user-guide/clusters/sizing.html#cluster-size-and-autoscaling) will be your best friend in managing the cost of the infrastructure.

**Arrive at Correct Cluster Size by Iterative Performance Testing**

*Impact: High*

It is impossible to predict the correct cluster size without developing the application because Spark and Azure Databricks use numerous techniques to improve cluster utilization. The broad approach you should follow for sizing is:

1. Develop on a medium sized cluster of 2-8 nodes, with VMs matched to workload class as explained earlier.
2. After meeting functional requirements, run end to end test on larger representative data while measuring CPU, memory and I/O used by the cluster at an aggregate level.
3. Optimize cluster to remove bottlenecks found in step 2
   * **CPU bound**: add more cores by adding more nodes
   * **Network bound**: use fewer, bigger SSD backed machines to reduce network size and improve remote read performance
   * **Disk I/O bound**: if jobs are spilling to disk, use VMs with more memory.

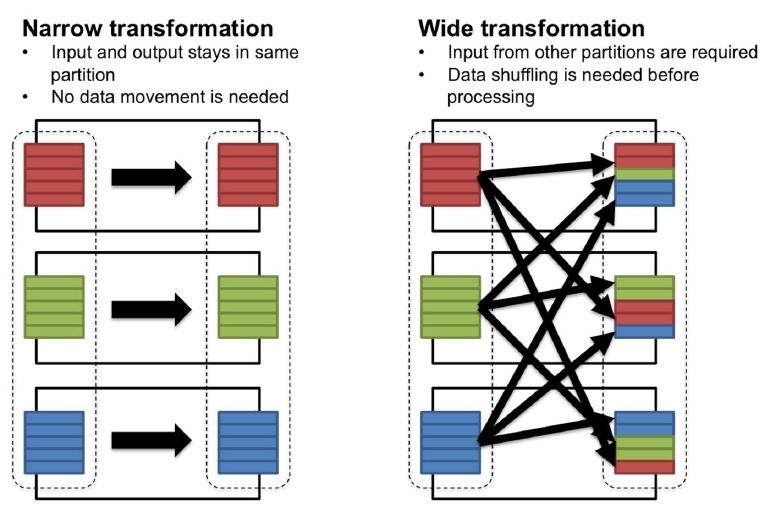
Repeat steps 2 and 3 by adding nodes and/or evaluating different VMs until all obvious bottlenecks have been addressed.

Performing these steps will help you to arrive at a baseline cluster size which can meet SLA on a subset of data. In theory, Spark jobs, like jobs on other Data Intensive frameworks (Hadoop) exhibit linear scaling. For example, if it takes 5 nodes to meet SLA on a 100TB dataset, and the production data is around 1PB, then prod cluster is likely going to be around 50 nodes in size. You can use this back of the envelope calculation as a first guess to do capacity planning. However, there are scenarios where Spark jobs don’t scale linearly. In some cases this is due to large amounts of shuffle adding an exponential synchronization cost (explained next), but there could be other reasons as well. Hence, to refine the first estimate and arrive at a more accurate node count we recommend repeating this process 3-4 times on increasingly larger data set sizes, say 5%, 10%, 15%, 30%, etc. The overall accuracy of the process depends on how closely the test data matches the live workload both in type and size.

**Tune Shuffle for Optimal Performance**

*Impact: High*

A shuffle occurs when we need to move data from one node to another in order to complete a stage. Depending on the type of transformation you are doing you may cause a shuffle to occur. This happens when all the executors require seeing all of the data in order to accurately perform the action. If the Job requires a wide transformation, you can expect the job to execute slower because all of the partitions need to be shuffled around in order to complete the job. Eg: Group by, Distinct.

[](https://github.com/Azure/AzureDatabricksBestPractices/blob/master/Figure7.PNG)

*Figure 7: Shuffle vs. no-shuffle*

You’ve got two control knobs of a shuffle you can use to optimize

* The number of partitions being shuffled: [SparkSnippet](https://github.com/Azure/AzureDatabricksBestPractices/blob/master/SparkSnippet.PNG)
* The amount of partitions that you can compute in parallel. + This is equal to the number of cores in a cluster.

These two determine the partition size, which we recommend should be in the Megabytes to 1 Gigabyte range. If your shuffle partitions are too small, you may be unnecessarily adding more tasks to the stage. But if they are too big, you may get bottlenecked by the network.

**Partition Your Data**

*Impact: High*

This is a broad Big Data best practice not limited to Azure Databricks, and we mention it here because it can notably impact the performance of Databricks jobs. Storing data in partitions allows you to take advantage of partition pruning and data skipping, two very important features which can avoid unnecessary data reads. Most of the time partitions will be on a date field but you should choose your partitioning field based on the predicates most often used by your queries. For example, if you’re always going to be filtering based on “Region,” then consider partitioning your data by region.

* Evenly distributed data across all partitions (date is the most common)
* 10s of GB per partition (~10 to ~50GB)
* Small data sets should not be partitioned
* Beware of over partitioning

**Running ADB Applications Smoothly: Guidelines on Observability and Monitoring**

***“Every program attempts to expand until it can read mail. Those programs which cannot so expand are replaced by ones which can.”*** Jamie Zawinski

By now we have covered planning for ADB deployments, provisioning Workspaces, selecting clusters, and deploying your applications on them. Now, let's talk about how to to monitor your Azure Databricks apps. These apps are rarely executed in isolation and need to be monitored along with a set of other services. Monitoring falls into four broad areas:

1. Resource utilization (CPU/Memory/Network) across an Azure Databricks cluster. This is referred to as VM metrics.
2. Spark metrics which enables monitoring of Spark applications to help uncover bottlenecks
3. Spark application logs which enables administrators/developers to query the logs, debug issues and investigate job run failures. This is specifically helpful to also understand exceptions across your workloads.
4. Application instrumentation which is native instrumentation that you add to your application for custom troubleshooting

For the purposes of this version of the document we will focus on (1). This is the most common ask from customers.

**Collect resource utilization metrics across Azure Databricks cluster in a Log Analytics workspace**

*Impact: Medium*

An important facet of monitoring is understanding the resource utilization in Azure Databricks clusters. You can also extend this to understanding utilization across all clusters in a workspace. This information is useful in arriving at the correct cluster and VM sizes. Each VM does have a set of limits (cores/disk throughput/network throughput) which play an important role in determining the performance profile of an Azure Databricks job. In order to get utilization metrics of an Azure Databricks cluster, you can stream the VM's metrics to an Azure Log Analytics Workspace (see Appendix A) by installing the Log Analytics Agent on each cluster node. Note: This could increase your cluster startup time by a few minutes.

**Querying VM metrics in Log Analytics once you have started the collection using the above document**

You can use Log analytics directly to query the Perf data. Here is an example of a query which charts out CPU for the VM’s in question for a specific cluster ID. See log analytics overview for further documentation on log analytics and query syntax.

%python

script = """

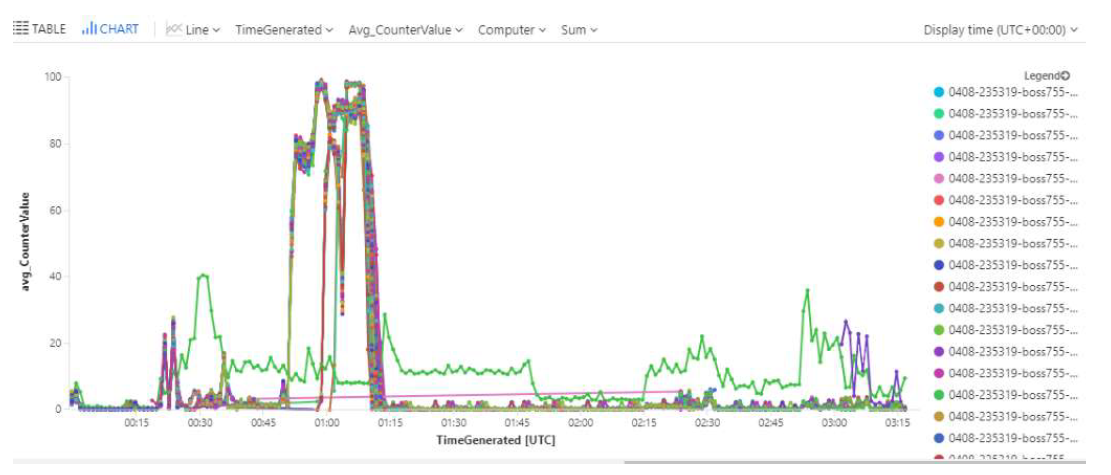
sed -i "s/^exit 101$/exit 0/" /usr/sbin/policy-rc.d

wget https://raw.githubusercontent.com/Microsoft/OMS-Agent-for-Linux/master/installer/scripts/onboard\_agent.sh && sh onboard\_agent.sh -w $YOUR\_ID -s $YOUR\_KEY -d opinsights.azure.com

"""

#save script to databricks file system so it can be loaded by VMs

dbutils.fs.put("/databricks/log\_init\_scripts/configure-omsagent.sh", script, True)

[](https://github.com/Azure/AzureDatabricksBestPractices/blob/master/Grafana.PNG)

You can also use Grafana to visualize your data from Log Analytics.

**Cost Management, Chargeback and Analysis**

This section will focus on Azure Databricks billing, tools to manage and analyze cost and how to charge back to the team.

**Azure Databricks Billing**

First, it is important to understand the different workloads and tiers available with Azure Databricks. Azure Databricks is available in 2 tiers – Standard and Premium. Premium Tier offers additional features on top of what is available in Standard tier. These include Role-based access control for notebooks, jobs, and tables, Audit logs, Azure AD conditional pass-through, conditional authentication and many more. Please refer to [Azure Databricks pricing](https://azure.microsoft.com/en-us/pricing/details/databricks/) for the complete list.

Both Premium and Standard tier come with 3 types of workload

* Jobs Compute (previously called Data Engineering)
* Jobs Light Compute (previously called Data Engineering Light)
* All-purpose Compute (previously called Data Analytics) The Jobs Compute and Jobs Light Compute make it easy for data engineers to build and execute jobs, and All-purpose make it easy for data scientists to explore, visualize, manipulate, and share data and insights interactively. Depending upon the use-case, one can also use All-purpose Compute for data engineering or automated scenarios especially if the incoming job rate is higher.

When you create an Azure Databricks workspace and spin up a cluster, below resources are consumed:

1. DBUs – A DBU is a unit of processing capability, billed on a per-second usage
2. Virtual Machines – These represent your Databricks clusters that run the Databricks Runtime
3. Public IP Addresses – These represent the IP Addresses consumed by the Virtual Machines when the cluster is running
4. Blob Storage – Each workspace comes with a default storage
5. Managed Disk
6. Bandwidth – Bandwidth charges for any data transfer

When you create an Azure Databricks workspace and spin up a cluster, below resources are consumed

* DBUs – A DBU is a unit of processing capability, billed on a per-second usage
* Virtual Machines – These represent your Databricks clusters that run the Databricks Runtime
* Public IP Addresses – These represent the IP Addresses consumed by the Virtual Machines when the cluster is running
* Blob Storage – Each workspace comes with a default storage
* Managed Disk
* Bandwidth – Bandwidth charges for any data transfer

| **Service or Resource** | **Pricing** |
| --- | --- |
| DBUs | [DBU pricing](https://azure.microsoft.com/en-us/pricing/details/databricks/) |
| VMs | [VM pricing](https://azure.microsoft.com/en-us/pricing/details/databricks/) |
| Public IP Addresses | [Public IP Addresses pricing](https://azure.microsoft.com/en-us/pricing/details/ip-addresses/) |
| Blob Storage | [Blob Storage pricing](https://azure.microsoft.com/en-us/pricing/details/storage/) |
| Managed Disk | [Managed Disk pricing](https://azure.microsoft.com/en-us/pricing/details/managed-disks/) |
| Bandwidth | [Bandwidth pricing](https://azure.microsoft.com/en-us/pricing/details/bandwidth/) |

In addition, if you use additional services as part of your end-2-end solution, such as Azure CosmosDB, or Azure Event Hub, then they are charged per their pricing plan.

There are 2 pricing plans for Azure Databricks DBUs:

1. Pay as you go – Pay for the DBUs as you use: Refer to the pricing page for the DBU prices based on the SKU. The DBU per hour price for different SKUs differs across Azure public cloud, Azure Gov and Azure China region.
2. Pre-purchase or Reservations – You can get up to 37% savings over pay-as-you-go DBU when you pre-purchase Azure Databricks Units (DBU) as Databricks Commit Units (DBCU) for either 1 or 3 years. A Databricks Commit Unit (DBCU) normalizes usage from Azure Databricks workloads and tiers into to a single purchase. Your DBU usage across those workloads and tiers will draw down from the Databricks Commit Units (DBCU) until they are exhausted, or the purchase term expires. The draw down rate will be equivalent to the price of the DBU, as per the table above.

Since, you are also billed for the VMs, you have both the above options for VMs as well:

1. Pay as you go
2. [Reservations](https://azure.microsoft.com/en-us/pricing/reserved-vm-instances/)

Please see few examples of a billing for Azure Databricks with Pay as you go:

Depending on the type of workload your cluster runs, you will either be charged for Jobs Compute, Jobs Light Compute, or All-purpose Compute workload. For example, if the cluster runs workloads triggered by the Databricks jobs scheduler, you will be charged for the Jobs Compute workload. If your cluster runs interactive features such as ad-hoc commands, you will be billed for All-purpose Compute workload.

Accordingly, the pricing will be dependent on below components

1. DBU SKU – DBU price based on the workload and tier
2. VM SKU – VM price based on the VM SKU
3. DBU Count – Each VM SKU has an associated DBU count. Example – D3v2 has DBU count of 0.75
4. Region
5. Duration

**Example 1: If you run Premium tier cluster for 100 hours in East US 2 with 10 DS13v2 instances, the billing would be the following for All-purpose Compute:**

* VM cost for 10 DS13v2 instances —100 hours x 10 instances x $0.598/hour = $598
* DBU cost for All-purpose Compute workload for 10 DS13v2 instances —100 hours x 10 instances x 2 DBU per node x $0.55/DBU = $1,100
* The total cost would therefore be $598 (VM Cost) + $1,100 (DBU Cost) = $1,698.

**Example 2: If you run Premium tier cluster for 100 hours in East US 2 with 10 DS13v2 instances, the billing would be the following for Jobs Compute workload:**

* VM cost for 10 DS13v2 instances —100 hours x 10 instances x $0.598/hour = $598
* DBU cost for Jobs Compute workload for 10 DS13v2 instances —100 hours x 10 instances x 2 DBU per node x $0.30/DBU = $600
* The total cost would therefore be $598 (VM Cost) + $600 (DBU Cost) = $1,198.

In addition to VM and DBU charges, there will be additional charges for managed disks, public IP address, bandwidth, or any other resource such as Azure Storage, Azure Cosmos DB depending on your application.

**Azure Databricks Trial**

If you are new to Azure Databricks, you can also use a Trial SKU that gives you free DBUs for Premium tier for 14 days. You will still need to pay for other resources like VM, Storage etc. that are consumed during this period. After the trial is over, you will need to start paying for the DBUs.

**Chargeback Scenarios**

There are 2 broad scenarios we have seen with respect to chargeback internal teams for sharing Databricks resources

1. Chargeback across a single Azure Databricks workspace: In this case, a single workspace is shared across multiple teams and user would like to chargeback the individual teams. Individual teams would use their own Databricks cluster and can be charged back at cluster level.
2. Chargeback across multiple Databricks workspace: In this case, teams use their own workspace and would like to chargeback at workspace level. To support these scenarios, Azure Databricks leverages Azure Tags so that the users can view the cost/usage for resources with tags. There are default tags that comes with the.

Please see below the default tags that are available with the resources:

| **Resources** | **Default Tags** |
| --- | --- |
| All-purpose Compute | Vendor, Creator, ClusterName, ClusterId |
| Jobs Compute or Jobs Light Compute | Vendor, Creator, ClusterName, ClusterId, RunName, JobId |
| Pool | Vendor, DatabricksInstancePoolId,DatabricksInstancePoolCreatorId |
| Resources created during workspace creation (Storage, Worker VNet, NSG) | application, databricks-environment |

In addition to the default tags, customers can add custom tags to the resources based on how they want to charge back. Both default and custom tags are displayed on Azure bills that allows one to chargeback by filtering resource usage based on tags.

1. [Cluster Tags](https://docs.microsoft.com/en-us/azure/databricks/clusters/configure#cluster-tags): You can create custom tags as key-value pairs when you create a cluster, and Azure Databricks applies these tags to underlying cluster resources – VMs, DBUs, Public IP Addresses, Disks.
2. [Pool Tags](https://docs.microsoft.com/en-us/azure/databricks/clusters/instance-pools/configure#--pool-tags): You can create custom tags as key-value pairs when you create a pool, and Azure Databricks applies these tags to underlying pool resources – VMs, Public IP Addresses, Disks. Pool-backed clusters inherit default and custom tags from the pool configuration.
3. [Workspace Tags](https://docs.microsoft.com/en-us/azure/databricks/administration-guide/account-settings/usage-detail-tags-azure): You can create custom tags as key-value pairs when you create an Azure Databricks workspaces. These tags apply to underlying resources within the workspace – VMs, DBUs, and others.

Please see below on how tags propagate for DBUs and VMs

1. Clusters created from pools

* DBU Tag = Workspace Tag + Pool Tag + Cluster Tag
* VM Tag = Workspace Tag + Pool Tag

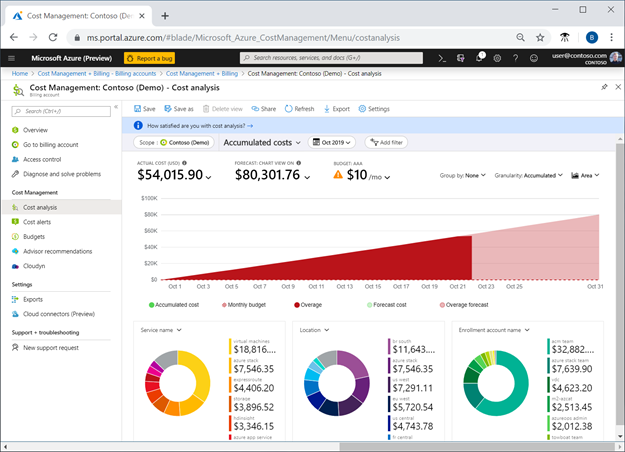
1. Clusters not from Pools

* DBU Tag = Workspace Tag + Cluster Tag
* VM Tag = Workspace Tag + Cluster Tag

These tags (default and custom) propagate to [Cost Analysis Reports](https://docs.microsoft.com/en-us/azure/cost-management-billing/costs/quick-acm-cost-analysis) that you can access in the Azure Portal. The below section will explain how to do cost/usage analysis using these tags.

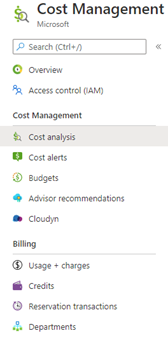
**Cost/Usage Analysis**

The Cost Analysis report is available under Cost Management within Azure Portal. Please refer to [Cost Management](https://docs.microsoft.com/en-us/azure/cost-management-billing/costs/quick-acm-cost-analysis)section to get a detailed overview on how to use Cost Management.

[](https://github.com/Azure/AzureDatabricksBestPractices/blob/master/Cost%20Management.png)

Below example is aimed at giving a quick start to get you going to do cost analysis for Azure Databricks. Below are the steps:

1. In Azure Portal, click on Cost Management + Billing
2. In Cost Management, click on Cost Analysis Tab

[](https://github.com/Azure/AzureDatabricksBestPractices/blob/master/Cost%20Management%20config.png)

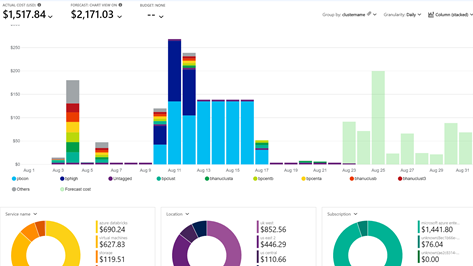
1. Choose the right billing scope that want report for and make sure the user has Cost Management Reader permission for the that scope.
2. Once selected, then you will see cost reports for all the Azure resources at that scope.
3. Post that you can create different reports by using the different options on the chart. For example, one of the reports you can create is

\* Chart option as Column (stacked)

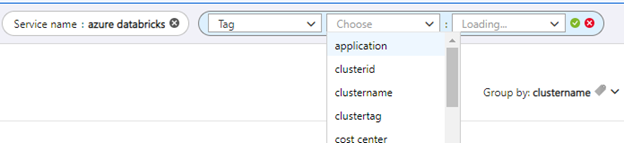
\* Granularity – Daily

\* Group by – Tag – Choose clustername or clustered

You will see something like below where it will show the distribution of cost on a daily basis for different clusters in your subscription or the scope that you chose in Step 3. You also have option to save this report and share it with your team.

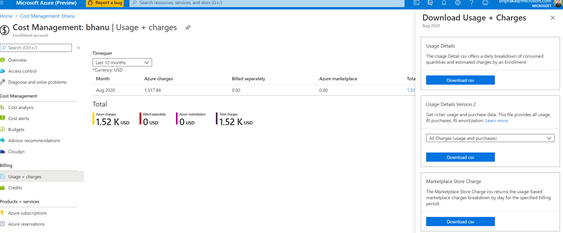
[](https://github.com/Azure/AzureDatabricksBestPractices/blob/master/Cost%20Management%20Report.png)

To chargeback, you can filter this report by using the tag option. For example, you can use default tag: Creator or can use own custom tag – Cost Center and chargeback based on that.

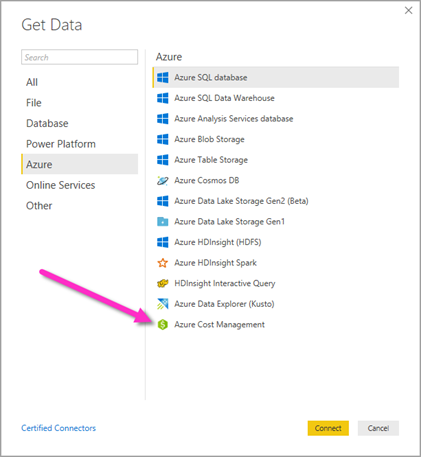
[](https://github.com/Azure/AzureDatabricksBestPractices/blob/master/Cost%20Management%20filter.png)

You also have option to consume this data from CSV or a native Power BI connector for Cost Management. Please see below:

1. To download this data to CSV, you can set export from Cost Management + Billing -> Usage + Charges and choose Usage Details Version 2 on the right. Refer [this](https://docs.microsoft.com/en-us/azure/cost-management-billing/reservations/understand-reserved-instance-usage-ea#download-the-usage-csv-file-with-new-data) for more details. Once downloaded, you can view the cost usage data and filter based on tags to chargeback. In the CSV, you can refer the Meter Name to get the Databricks workload consumed. In addition, this is how the other fields are represented for meters related to Azure Databricks.
   * Quantity = Number of Virtual Machines x Number of hours x DBU count
   * Effective Price = DBU price based on the SKU
   * Cost = Quantity x Effective Price

[](https://github.com/Azure/AzureDatabricksBestPractices/blob/master/Cost%20Management%20export.png)

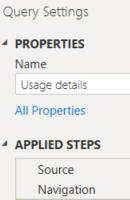
1. There is a native [Cost Management Connector](https://docs.microsoft.com/en-us/power-bi/connect-data/desktop-connect-azure-cost-management) in Power BI that allows one to make powerful, customized visualization and cost/usage reports.

[](https://github.com/Azure/AzureDatabricksBestPractices/blob/master/Cost%20Management%20connector.png)

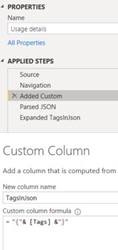
Once you connect, you can create various rich reports easily like below by choosing the right fields from the table.

Tip: To filter on tags, you will need to parse the json in Power BI. To do that, follow these steps:

1. Go to "Query Editor"
2. Select the "Usage Details" table
3. On the right side the "Properties" tab shows the steps as

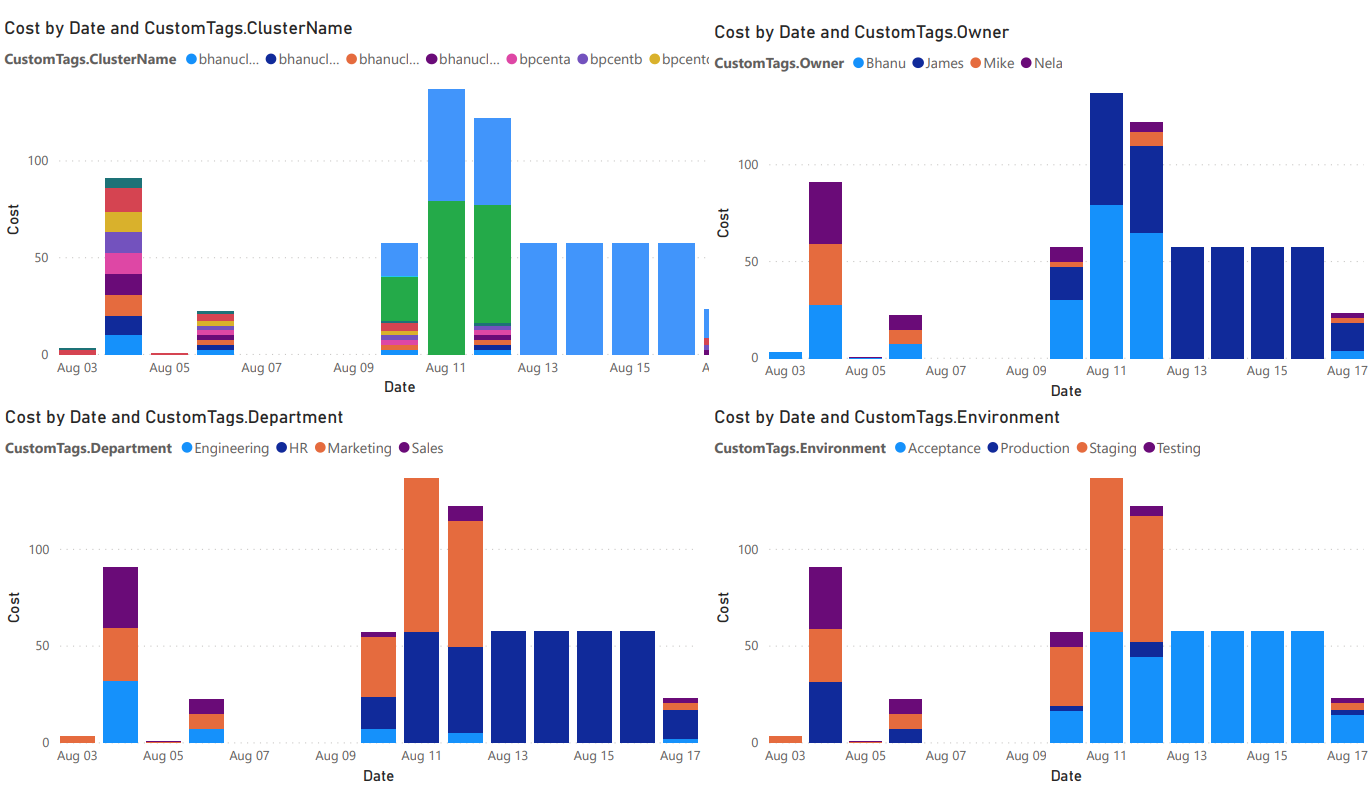
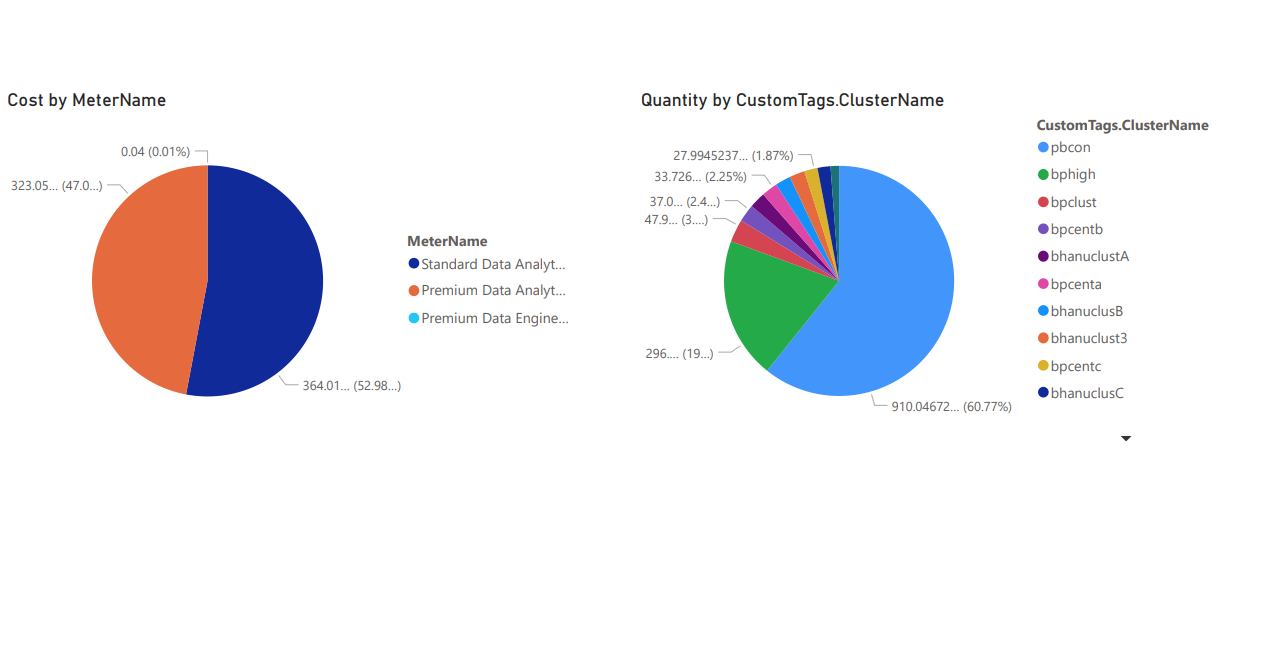
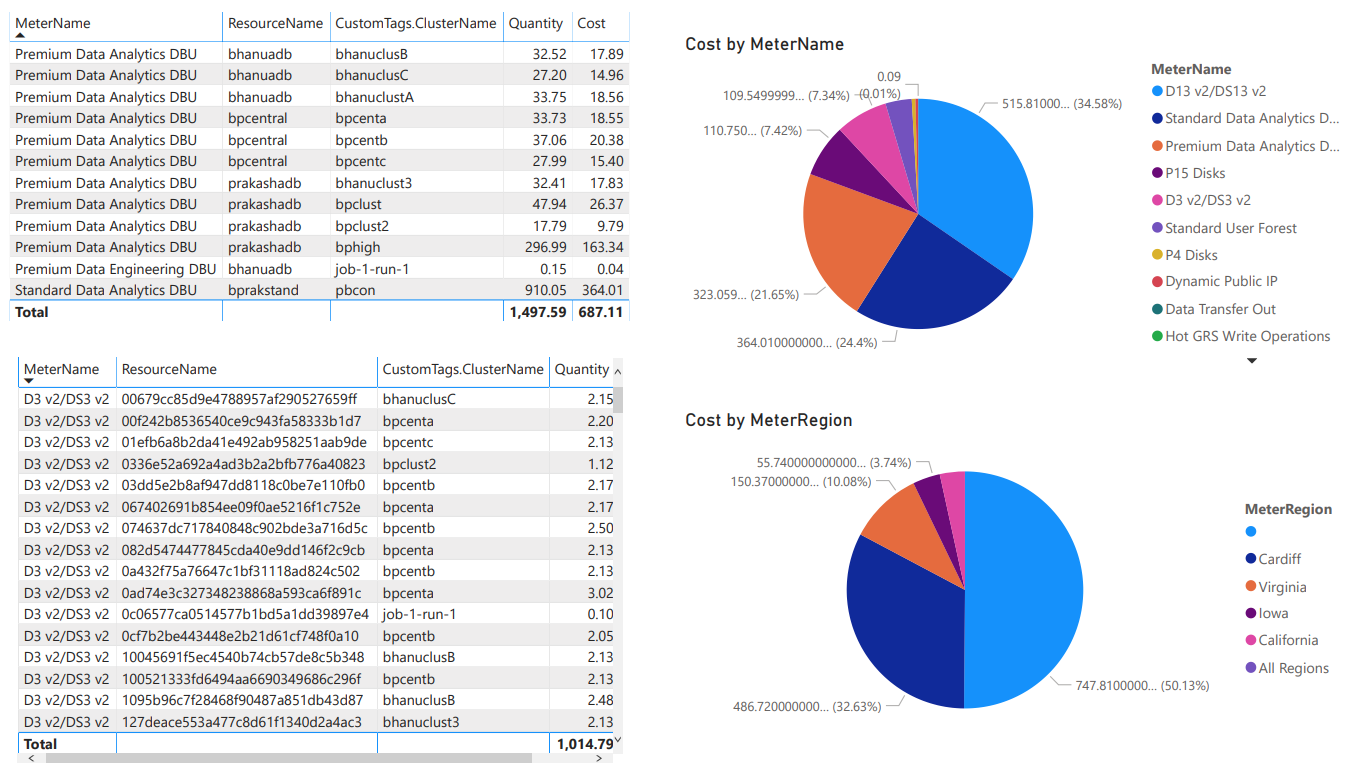
[](https://github.com/Azure/AzureDatabricksBestPractices/blob/master/Cost%20Management%20PBI%20config.png)

1. From the menu bar go to "Add column" -> "Add custom column"
2. Name the column and enter the following text in the query = "{"& [Tags] & "}"

[](https://github.com/Azure/AzureDatabricksBestPractices/blob/master/Cost%20Management%20PBI%20config2.jpg)

1. This will create a new column of "tags" in the json format.
2. Now user can transform it as expand it. You can then use the different tags as columns that you can use in a report.

Please see some of the common views created easily using this connector.

* Cost Report breakdown by Resource Group, Tags, MeterName [](https://github.com/Azure/AzureDatabricksBestPractices/blob/master/Cost%20Management%20PBI%20report%201.png)
* Cost Report breakdown by Cluster, and custom tags [](https://github.com/Azure/AzureDatabricksBestPractices/blob/master/Cost%20Management%20PBI%20report2.PNG)
* Cost Report breakdown by Cluster and Metername in pie chart [](https://github.com/Azure/AzureDatabricksBestPractices/blob/master/Cost%20Management%20PBI%20report3.PNG)
* Cost Report breakdown by with resource group and cluster including quantity [](https://github.com/Azure/AzureDatabricksBestPractices/blob/master/Cost%20Management%20PBI%20report4.PNG)

**Pricing difference for Regions**

Please refer to [Azure Databricks pricing page](https://azure.microsoft.com/en-us/pricing/details/databricks/) to get the pricing for DBU SKU and pricing discount based on Reservations. There are certain differences to consider 1.The DBU prices are different for Azure public cloud and other regions such as Azure Gov 2.The pre-purchase plan prices are different for Azure public cloud and Azure Gov

**Known Issues/Limitations**

1. Tag change propagation at workspace level takes up to ~1 hour to apply to resources under Managed resource group.
2. Tag change propagation at workspace level requires cluster restart for existing running cluster, or pool expansion
3. Cost Management at parent resource group won’t show Managed RG resources consumption
4. Cost Management role assignments are not possible at Managed RG level. User today must have role assignment at parent resource group level or above (i.e. subscription) to show managed RG consumption
5. For clusters created from pool, only workspace tags and pool tags are propagated to the VMs
6. Tag keys and values can contain only characters from ISO 8859-1 set
7. Custom tag gets prefixed with x\_ when it conflicts with default tag
8. Max of 50 tags can be assigned to Azure resource