Azure Databricks Best Practices

databricks

Agenda

- Workspace Admin Best Practices
- Security Best Practices
- Tools & Integration Best Practices
- Databricks Runtime Best Practices
- HA & DR Best Practices
- Cluster Best Practices

Workspace Admin Best Practices

- Create different workspaces by different department / business team / data tier, and per environment (dev, qa, prod) - across relevant Azure subscriptions
- Define workspace level tags which propagate to initially provisioned resources in managed resource group (Tags could also propagate from parent resource group)
- Use ARM templates (search "databricks") to have a more managed way of deploying the workspaces - whether via CLI, powershell or some SDK
- Create relevant groups of users - using Group REST API or by using AAD Group Sync with SCIM

Security Best Practices

- Do not store any production data on DBFS (use it only for toy / experimental datasets).
- Configure encryption-at-rest for Blob Storage and ADLS, preferably by using customer-managed keys in Azure Key Vault.
- Use Secrets with Azure Key Vault backend to obfuscate passwords and keys in notebooks.
- Prefer to use ADLS credential passthrough over Table ACLs (if possible).
- Configure access control for Databricks-native resources (clusters, notebooks, jobs etc.)
- Deploy workspace in your VNET to enable networking customizations.
- Configure Audit Logs to monitor the activity in a workspace.

Tools & Integration Best Practices

- Use Azure Data Factory to orchestrate pipelines / workflows (or something like Airflow).
- Connect your IDE or custom applications to Azure Databricks clusters using DB-Connect (Private Preview).
- Sync notebooks with Azure Devops for seamless version control.
- Use Databricks CLI for CI / CD from relevant enterprise tools/products, or to integrate with other systems like on-prem SCM or Library Repos etc.
- Use Library Utilities to install python libraries scoped at notebook level (cluster-scoped libraries may make more sense in certain cases).
- Use Init Scripts to do custom installs at cluster level.

Databricks Runtime Best Practices

- Use Delta wherever you can, to get the best performance and reliability for your big data workloads, and to create no-fuss multi-step data pipelines.
- Use Machine Learning Runtime for working with the latest ML/DL libraries (including HorovodRunner for distributed DL).
- Use Delta Cache for accelerating reads from Blob Storage or ADLS.
- Use ABS-AQS connector for structured streaming when working with consistent rate of incoming files on Blob Storage.
- Turn on Databricks Advisor for automated tips on how to optimize workload processing.

HA and DR Best Practices

- Deploy Azure Databricks in two paired azure regions, ideally mapped to different control plane regions.
  - E.g. East US2 and West US2 will map to different control planes
  - Whereas West and North Europe will map to same control plane
- Use Azure Traffic Manager to load balance and distribute API requests between two deployments, when the platform is primarily being used in a backend non-interactive mode.
- Design to honor API and other limits of the platform.
  - Max API calls/ hr = 1500
  - Jobs per hour per workspace = 1000
  - Maximum concurrent Notebooks per cluster = 145

Cluster Best Practices

- Use autoscaling and auto-termination wherever applicable (e.g. auto-termination doesn't make sense if you need a cluster for data analysis by multiple users almost through the day, etc.).
- Use latest Databricks Runtime version to take advantage of latest performance & other optimizations (applicable in most cases, though not all).
- Use High-concurrency cluster mode for data analysis by a team of users via notebooks or a BI tool, or if you want to enforce data protection via Table ACLs or ADLS Passthrough.
- Use cluster tags for project / team based chargeback.

Cluster Best Practices Contd..

- Use Spark config tab if certain tuning would make sense for a specific workload (like config to use broadcast join).
- Use Event Log and Spark UI to see how different queries / workload executions perform, and what affect those have on a cluster's health.
- Configure Cluster Log Delivery
- Use Cluster ACLs to configure what each user or a group of users are allowed to do.
- Refer this blog by a customer, which more or less mentions what we've covered here. Rest is really workload dependent where it requires evidence-based tuning.

Appendix - Choosing the instance type

Different Azure Instance Types

Compute Optimized

- Fs
  - Haswell processor (Skylake not supported yet)
  - 1 core ~ 2GB RAM
  - SSD Storage: 1 core ~ 16GB
- H
  - High-performance
  - 1 core ~ 7GB RAM
  - SSD Storage: 1 core ~ 125GB

Memory Optimized

- DSv2
  - Haswell processor
  - 1 core ~ 7GB RAM
  - SSD Storage: 1 core ~ 14 GB
- ESv3
  - High-performance (Broadwell processor)
  - 1 core ~ 8GB RAM
  - SSD Storage: 1 core ~ 16GB

Storage Optimized

- L
  - 1 core ~ 8GB RAM
  - SSD Storage: 1 core ~ 170GB
  - Price : .156

General Purpose

- DSv2 and DSv3
  - DSv2 - 1 core ~ 3.5GB RAM
  - DSv3 - 1 core ~ 4GB RAM
  - SSD Storage:
    - DSv2 - 1 core ~ 7GB
    - DSv3 - 1 core ~ 8GB

Cluster Sizing Starting Points

Rules of Thumb

- Fewer big instances > more small instances
  - Reduce network shuffle; Databricks has 1 executor / machine
  - Applies to batch ETL mainly (for streaming, one could start with smaller instances depending on complexity of transformation)
  - Not set in stone, and reverse would make sense in many cases - so sizing exercise matters
- Size based on the number of tasks initially, tweak later
  - Run the job with a small cluster to get idea of # of tasks (use 2-3x tasks per core for base sizing)
- Choose based on workload (Probably start with F-series or DSv2):
  - ETL with full file scans and no data reuse - F / DSv2
  - ML workload with data caching - DSv2 / F
  - Data Analysis - L
  - Streaming - F

How do we tweak these?

Workload requires caching (like machine learning)

- Look at the Storage tab in Spark UI to see if the entirety of the training dataset is cached
  - Fully cached with room to spare -> less instances
  - Partially cached
    - Almost completely cached? -> Increase the cluster size
    - Not even close to cached -> Consider L series or DSv2 memory-optimized
      - Check to see if persist is MEMORY_ONLY, or MEMORY_AND_DISK
      - Spill to disk with SSD isn't so bad
- Still not good enough? Follow the steps in the next section

How do we tweak these?

ETL and Analytic Workloads

- Are we compute bound?
  - Check CPU Usage (Ganglia metrics to come to Azure Databricks soon)
  - Only way to make faster is more cores
- Are we network bound?
  - Check for high spikes before compute heavy steps
  - Use bigger/fewer machines to reduce the shuffle
  - Use an ssd backed instance for faster remote reads
- Are we spilling a ton?
  - Check Spark SQL tab for spill (pre-agg before shuffles are common to spill)
    - Use L-series
    - Or use more memory