Sterling Architecture Refresh

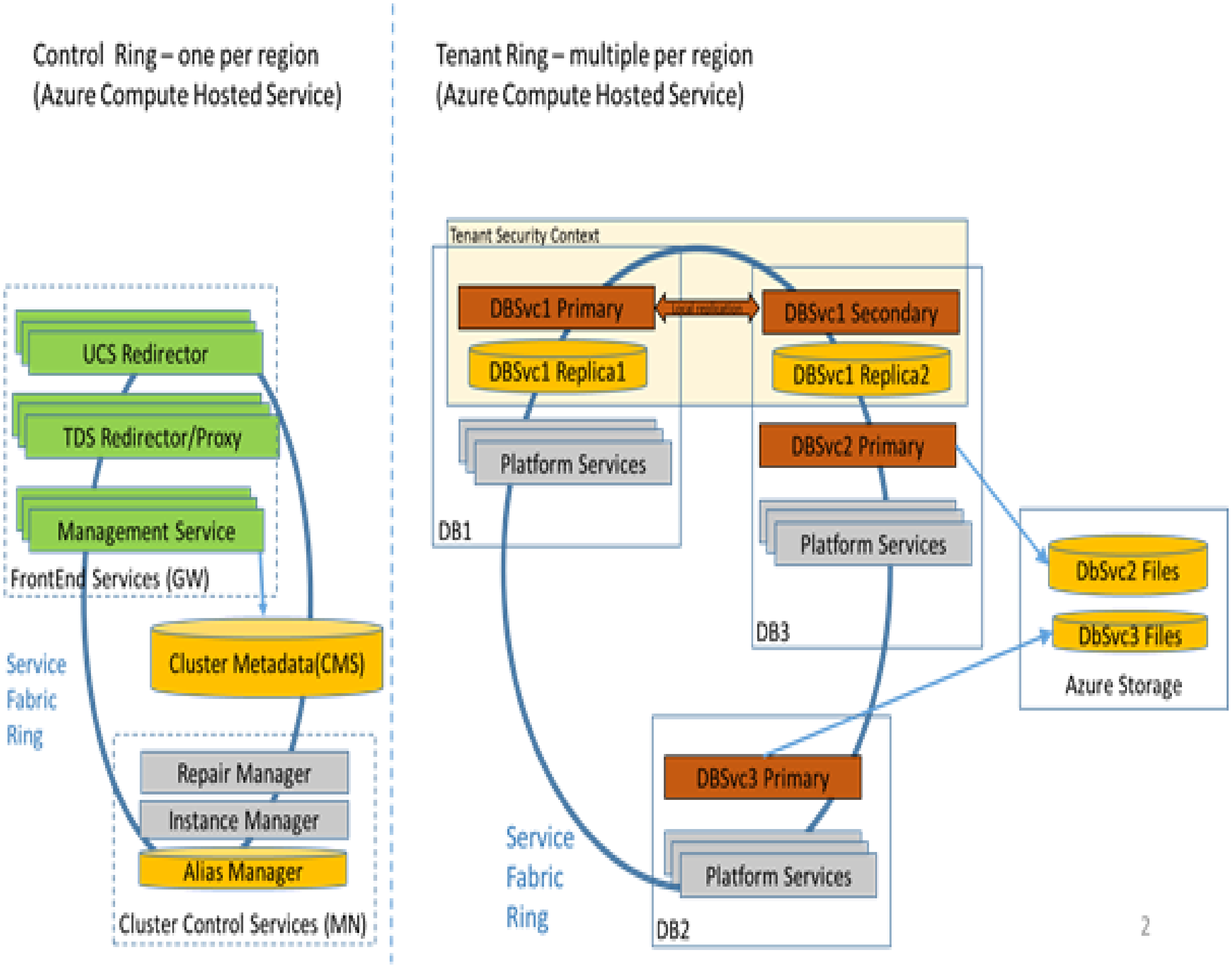
Last updated by | Charlene Wang | Apr 12, 2021 at 11:27 PM PDT

Contents

Sterling Architecture Refresh

Sterling Components

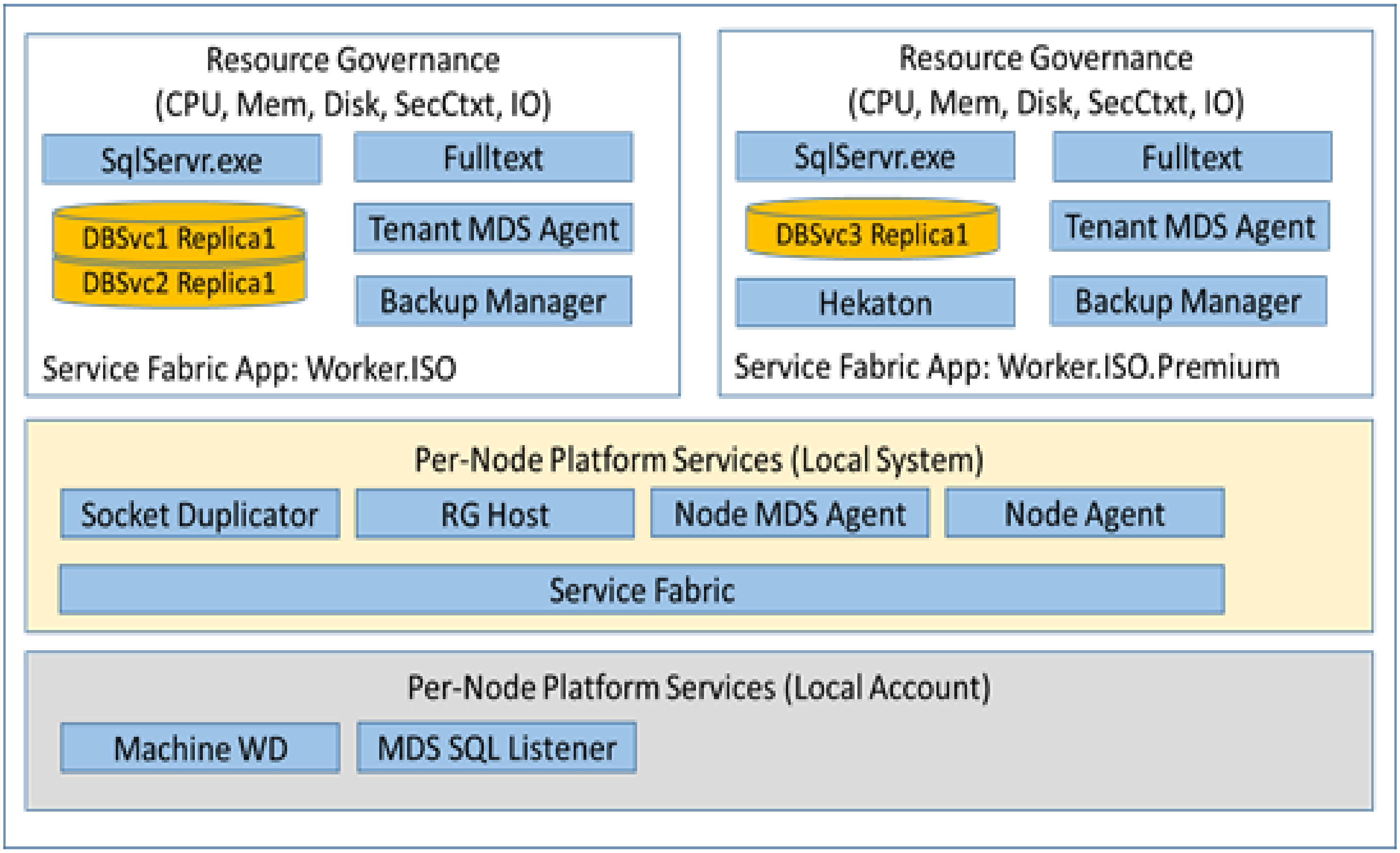
Sterling components are hosted in a Service Fabric Ring which in turn hosts multiple Service Fabric application instances, all composing to form a Sterling service. The Sterling service is divided into one Control Ring and multiple Tenant Rings.



Tenant Ring

In Sterling, a tenant ring is composed by definitive number of nodes that host the Application Services, representing per-tenant Database Services hosting a group of databases in the same server, and the Sterling Platform Services, a group of per-node services that help in administering the database instances running on the node.

The Database Services are the ones that have the data and may have secondary replicas. The Database Services move around between nodes on failover. The Platform Services do not have state and are tied to a node for their life.



- Database Application instance (Worker.ISO.*) runs the sqlservr.exe process that provides the SQL capabilities to the single tenant. The process can host one or more database services, each representing a logical database to the user. In addition to the SQL process, the application runs:
  - MDS Agent for uploading tenant-specific telemetry to the MDS store.
  - Full Text Search process and possibly other auxiliary SQL processes that provide full surface area of SQL functionality.
  - Backup Manager that is responsible for taking data backups or restoring previous backups to new databases.

  The code running inside the Database Application instance is specific to the service provided to the user. Today we have two versions of SQL DB applications – **Worker.ISO** (for Basic and Standard editions) and **Worker.ISO.Premium** (for Premium edition), as well as **Worker.ISO.DW** for the SQL DW databases. Each application type can have a different mix of hosted processes and different configuration.

- Per-Node Platform Services are running one instance of an application per node. The following applications are hosted on a backend DB node:
  - Socket Duplicator – listens for incoming connection requests on one port, then duplicates socket/SSL context to a per-DbSvc port.
  - RG Host – responsible for implementing resource containment policy (NtJobs) for all DbSvc applications in that node. Responsible for providing low privilege unique accounts to each tenant.
  - Dynamic Activator: Responsible for starting and stopping DbSvc instances based on various criteria (incoming requests, length of non-use, etc.) to achieve COGS, startup-time, etc goals.
  - Node Agent – relays requests for administrative operations from Management Service in Control Ring over to individual DBSvs instances on the node. It can connect with full admin privileges to each

instance and run pre-configured SQL statements.
- MDS agents – various MDS agents for collecting node-level information and pushing it to MDS.

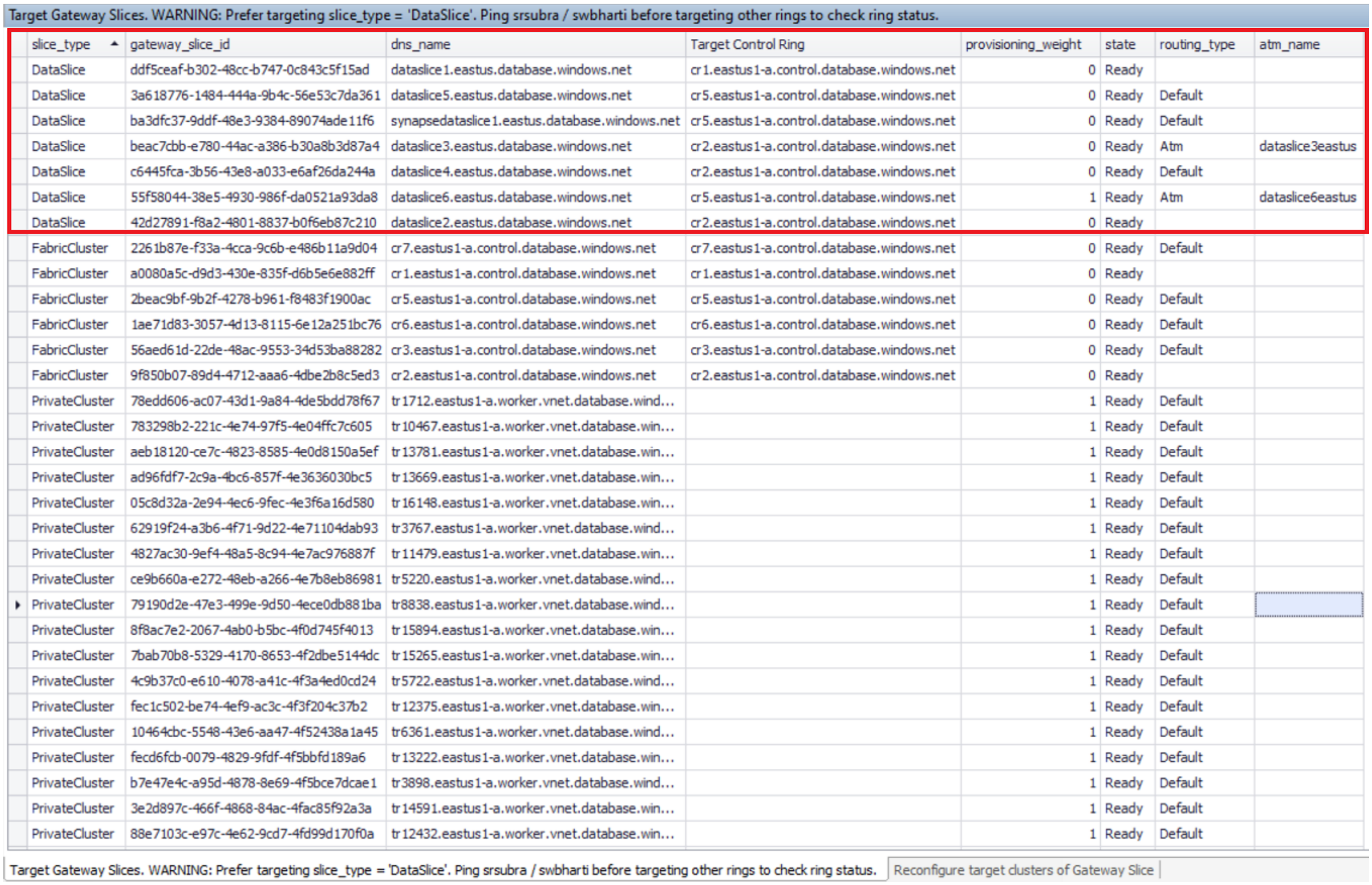
Control Ring

Control Ring is fully autonomous and can run independently of Tenant Rings. On the opposite, Tenant Rings need the Control Ring to function as part of Sterling cluster.

Azure SQL Database control rings are running hot in many regions causing some downtimes for customers, so there is another introduced load balancing concept **data slice** to handle the connections. Multiple control rings are combined under a dataslice which sits behind an azure traffic manager. This offers us two benefits:

1. Having two different CRs (each belonging to a different AZ whenever possible) gives us better tolerance in case one of the CRs is in trouble. In that case the second CR linked to the dataslice can take over all the traffic while the first CR recovers. The clients on their end should notice no difference while this switch happens.
2. Using an Azure Traffic Managers lets us better utilize the capacity of both the CRs. We can control the percentage of total logins that land on the dataslice that will be serviced by each CR. To understand better about how traffic manager works, please refer to following link: https://docs.microsoft.com/en-us/azure/traffic-manager/traffic-manager-how-it-works

There might be mulitple data slice per Azure region and under each data slice, there might be multiple control rings. You can view the slice info in each region using gateway slice.xts.

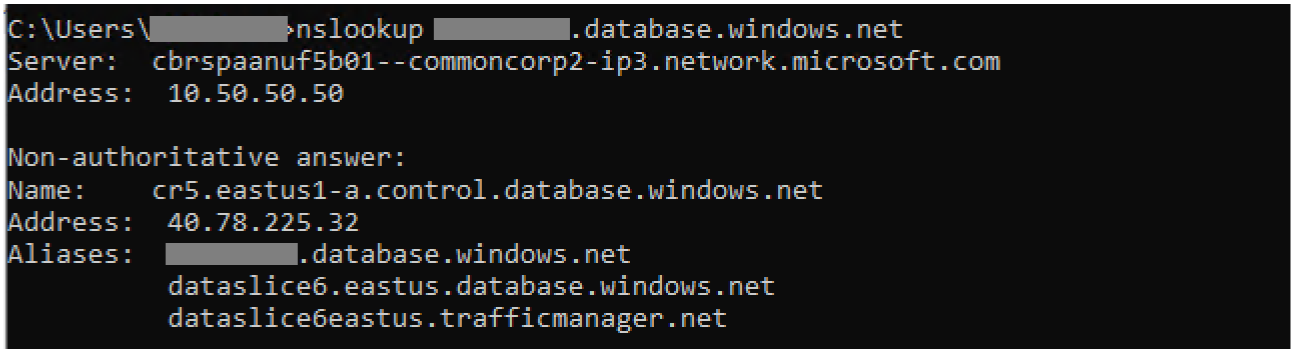


Target Gateway Slices. WARNING: Prefer targeting slice_type = 'DataSlice'. Ping srsubra / swbharti before targeting other rings to check ring status.

slice_type	gateway_slice_id	dns_name	Target Control Ring	provisioning_weight	state	routing_type	atm_name
DataSlice	ddf5ceaf-b302-48cc-b747-0c843c5f15ad	dataslice1.eastus.database.windows.net	cr1.eastus1-a.control.database.windows.net	0	Ready		
DataSlice	3a618776-1484-444a-9b4c-56e53c7da361	dataslice5.eastus.database.windows.net	cr5.eastus1-a.control.database.windows.net	0	Ready	Default	
DataSlice	ba3dfc37-9ddf-48e3-9384-89074ade11f6	synapsedataslice1.eastus.database.windows.net	cr5.eastus1-a.control.database.windows.net	0	Ready	Default	
DataSlice	beac7cbb-e780-44ac-a386-b30a8b3d87a4	dataslice3.eastus.database.windows.net	cr2.eastus1-a.control.database.windows.net	0	Ready	Atm	dataslice3eastus
DataSlice	c6445fca-3b56-43e8-a033-e6af26da244a	dataslice4.eastus.database.windows.net	cr2.eastus1-a.control.database.windows.net	0	Ready	Default	
DataSlice	55f58044-38e5-4930-986f-da0521a93da8	dataslice6.eastus.database.windows.net	cr5.eastus1-a.control.database.windows.net	1	Ready	Atm	dataslice6eastus
DataSlice	42d27891-f8a2-4801-8837-b0f6eb87c210	dataslice2.eastus.database.windows.net	cr2.eastus1-a.control.database.windows.net	0	Ready		
FabricCluster	2261b87e-f33a-4cca-9c6b-e486b11a9d04	cr7.eastus1-a.control.database.windows.net	cr7.eastus1-a.control.database.windows.net	0	Ready	Default	
FabricCluster	a0080a5c-d9d3-430e-835f-d6b5e6e882ff	cr1.eastus1-a.control.database.windows.net	cr1.eastus1-a.control.database.windows.net	0	Ready		
FabricCluster	2beac9bf-9b2f-4278-b961-f8483f1900ac	cr5.eastus1-a.control.database.windows.net	cr5.eastus1-a.control.database.windows.net	0	Ready	Default	
FabricCluster	1ae71d83-3057-4d13-8115-6e12a251bc76	cr6.eastus1-a.control.database.windows.net	cr6.eastus1-a.control.database.windows.net	0	Ready	Default	
FabricCluster	56aed61d-22de-48ac-9553-34d53ba88282	cr3.eastus1-a.control.database.windows.net	cr3.eastus1-a.control.database.windows.net	0	Ready	Default	
FabricCluster	9f850b07-89d4-4712-aaa6-4dbe2b8c5ed3	cr2.eastus1-a.control.database.windows.net	cr2.eastus1-a.control.database.windows.net	0	Ready		
PrivateCluster	78edd606-ac07-43d1-9a84-4de5bdd78f67	tr1712.eastus1-a.worker.vnet.database.wind...		1	Ready	Default	
PrivateCluster	783298b2-221c-4e74-97f5-4e04ffc7c605	tr10467.eastus1-a.worker.vnet.database.win...		1	Ready	Default	
PrivateCluster	aeb18120-ce7c-4823-8585-4e0d8150a5ef	tr13781.eastus1-a.worker.vnet.database.win...		1	Ready	Default	
PrivateCluster	ad96fdf7-2c9a-4bc6-857f-4e3636030bc5	tr13669.eastus1-a.worker.vnet.database.win...		1	Ready	Default	
PrivateCluster	05c8d32a-2e94-4ec6-9fec-4e3f6a16d580	tr16148.eastus1-a.worker.vnet.database.win...		1	Ready	Default	
PrivateCluster	62919f24-a3b6-4f71-9d22-4e71104dab93	tr3767.eastus1-a.worker.vnet.database.wind...		1	Ready	Default	
PrivateCluster	4827ac30-9ef4-48a5-8c94-4e7ac976887f	tr11479.eastus1-a.worker.vnet.database.win...		1	Ready	Default	
PrivateCluster	ce9b660a-e272-48eb-a266-4e7b8eb86981	tr5220.eastus1-a.worker.vnet.database.wind...		1	Ready	Default	
PrivateCluster	79190d2e-47e3-499e-9d50-4ece0db881ba	tr8838.eastus1-a.worker.vnet.database.wind...		1	Ready	Default	
PrivateCluster	8f8ac7e2-2067-4ab0-b5bc-4f0d745f4013	tr15894.eastus1-a.worker.vnet.database.win...		1	Ready	Default	
PrivateCluster	7bab70b8-5329-4170-8653-4f2dbe5144dc	tr15265.eastus1-a.worker.vnet.database.win...		1	Ready	Default	
PrivateCluster	4c9b37c0-e610-4078-a41c-4f3a4ed0cd24	tr5722.eastus1-a.worker.vnet.database.wind...		1	Ready	Default	
PrivateCluster	fec1c502-be74-4ef9-ac3c-4f3f204c37b2	tr12375.eastus1-a.worker.vnet.database.win...		1	Ready	Default	
PrivateCluster	10464cbc-5548-43e6-aa47-4f52438a1a45	tr6361.eastus1-a.worker.vnet.database.wind...		1	Ready	Default	
PrivateCluster	fecd6fcb-0079-4829-9fdf-4f5bbfd189a6	tr13222.eastus1-a.worker.vnet.database.win...		1	Ready	Default	
PrivateCluster	b7e47e4c-a95d-4878-8e69-4f5bce7dcae1	tr3898.eastus1-a.worker.vnet.database.wind...		1	Ready	Default	
PrivateCluster	3e2d897c-466f-4868-84ac-4fac85f92a3a	tr14591.eastus1-a.worker.vnet.database.win...		1	Ready	Default	
PrivateCluster	88e7103c-e97c-4e62-9cd7-4fd99d170f0a	tr12432.eastus1-a.worker.vnet.database.win...		1	Ready	Default	

Target Gateway Slices. WARNING: Prefer targeting slice_type = 'DataSlice'. Ping srsubra / swbharti before targeting other rings to check ring status.

Each logical server is mapped to one data slice and depends on the load, data slice may route the traffic to different control ring binding to that data slice. You can find the DNS resolution through nslookup.



```
C:\Users\[redacted]>nslookup [redacted].database.windows.net
Server:  cbrspaanuf5b01--commoncorp2-ip3.network.microsoft.com
Address:  10.50.50.50

Non-authoritative answer:
Name:    cr5.eastus1-a.control.database.windows.net
Address:  40.78.225.32
Aliases:  [redacted].database.windows.net
          dataslice6.eastus.database.windows.net
          dataslice6eastus.trafficmanager.net
```

Control Rings holds information about all entities (subscriptions, servers, databases, etc) present in that region and controls traffic to those entities.

Control Ring registers endpoints that listen on all protocols that can be used to communicate with hosted services (Azure SQL DB, Azure SQL DW) and receives traffic routed through DNS entry registered for the region.

The traffic coming to Control Ring can be described in two categories:

- Data Plane – all requests to access and manipulate data stored in the cluster. This traffic is using protocols native to the engines storying the data – for SQL DB and SQL DW those are currently TDS and UCS.
- Control Plane – all requests to access and manage objects (entities) in the cluster. This traffic is using standardized protocols and APIs across Azure, such that all Azure services can be managed through a coherent set of APIs and client tools and portals. The Control Plane traffic uses HTTP(s) and REST APIs.

The components in the Control Ring are running on two sets of nodes and subsequently can be grouped in the two sets:

- Front-end services, running on GW nodes:
    - TDS Proxy/Redirector – listens to incoming connection and redirects or proxies them (depending on the source of the connection) to the appropriate DB Svc in one of the Tenant Rings. The TDS protocol is specific to SQL Server.
    - UCS Redirector – responsible for marshalling connections for replication protocol (HADRON) between multiple clusters. This is used to implement Active Geo-Replication.
    - Management Service – responsible for implementing Resource Provider endpoints that plug into overarching Azure ARM Frontend t/hat aggregates all Azure APIs. The Resource Provider implements all REST APIs used to expose Azure SQL DB management functionality to the customers. In addition to that Management Service provides additional endpoints for internal Cluster Management (allowing our engineers to manage the cluster and all resources within it), and for Deployment Rollouts (allowing our Deployment Automation to coordinate the code/config deployments to the cluster). Some of those additional functions may be pulled out into separate services in the future.
- Cluster control services, running on MN nodes:
    - Cluster Metadata Store (CMS) – is a central database hosted in Control Ring and backed by remote storage in Azure Storage. It holds all metadata about the cluster (“source of the truth”) and is used to ensure atomicity and durability of all Control Plane workflows orchestrated by Management Service.
    - Alias Store/Manager – another database used to store information about location of all DB Svc in the tenant ring. It is updated as the Service Fabric moves those services around. This data store is critical

for the connectivity workflows.
    - Instance Manager – is an internal head-less service that manages the CMS server, controls updates to it and monitors its health. It is also used in bootstrapping the Control Ring.
    - Repair Manager – responsible for examining health of the nodes in the cluster and reacting to failures at SQL level that may be undetected by Service Fabric or Azure layers.

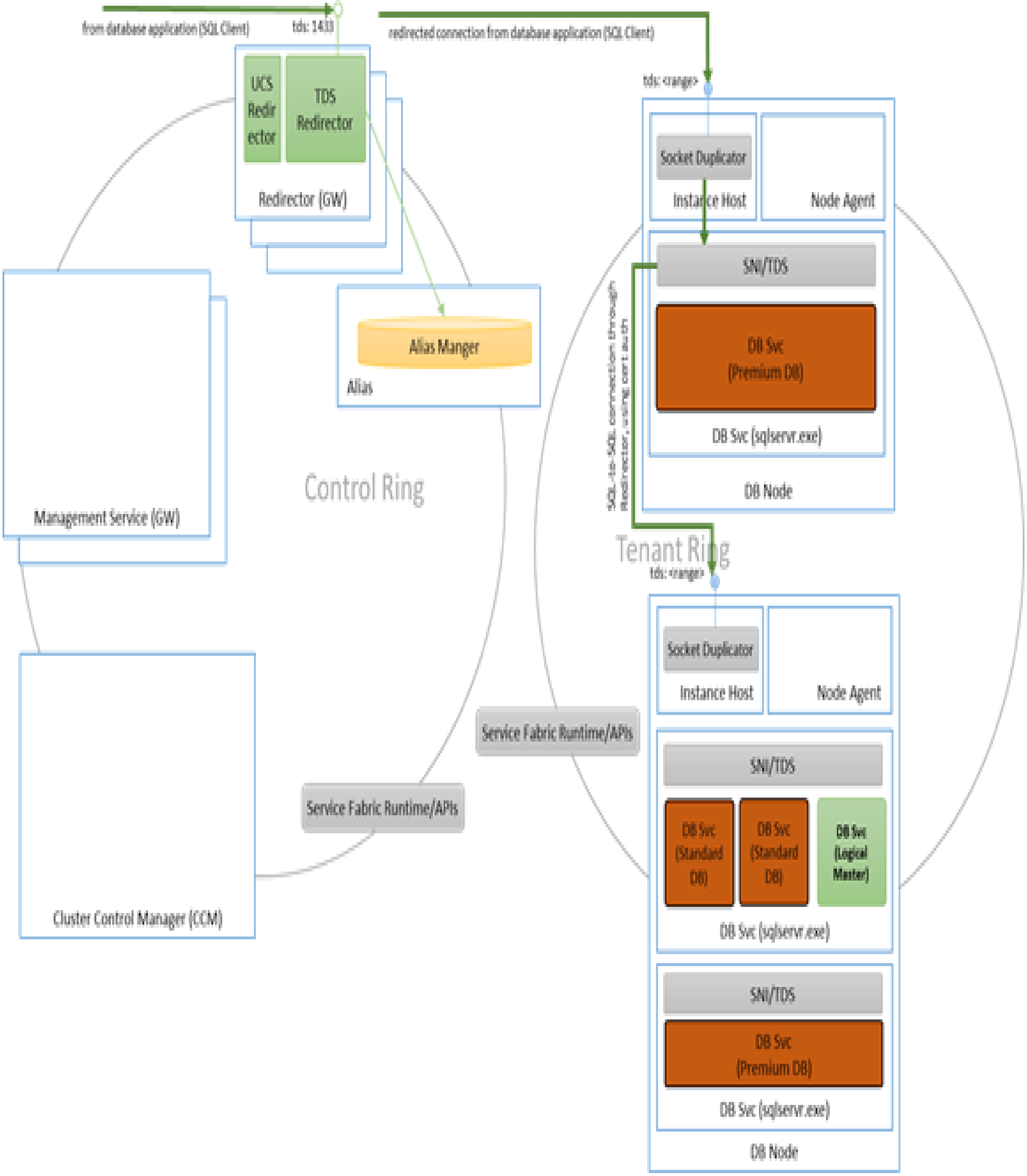
High Availability

- **Basic/Standard tiers** keep the data files remotely (outside of the node running the server process) on Azure Storage. The data is stored in BLOB containers, with one BLOB hosting one file. SQL Server code has been modified to talk directly to Azure Storage using REST APIs for page get/put operations. This could have been done because per-page BLOB access aligns well with how SQL Server uses I/O system with local files.
    - The Azure Storage internally ensures high availability of the stored data by hosting multiple copies of the same BLOB locally in each data center. SQL DB service relies on that availability guarantee and runs only a single instance of sqlservr.exe process in the Tenant Ring. If that service instance fails, Service Fabric starts automatically another one on another node and the new instance connects to the same storage container and picks up where the previous one left.
    - Depending on the Performance Tier (Basic, S0, S1, S2, S3), we use either Standard Storage (for lower tiers) or Premium Storage (for higher tiers). Premium Storage offers lower latencies and better performance guarantees and performance isolation between tenants.
- **Premium tier** databases keep the data locally on SSD drives in the compute nodes. The service running a Premium database is configured to start 4 separate replicas (primary and 3 secondary replicas). Service Fabric coordinates the creation and placement of the replicas and establishes the replication links between them. SQL Server uses the HARDON replication mechanism to synchronize data between the replicas. Once replication is established, each transaction in the DB needs to be committed at a minimum number of replicas (quorum) before the success is communicated to the caller. If the primary replica fails, Service Fabric will elect a new primary among the existing secondary replicas, and the new connections will be routed to it. Because the replicas are synchronized, the committed changes continue to be durable.
    - Because Premium tier uses the local SSD drives it has the highest performance characteristics of all the tiers.

Connectivity

- **Redirection** – the connection is received by the GW node in Control Ring first. The Redirector service uses the Alias Store/Manager to find the present location of the DB Svc in one of the tenant rings. Once that is established the location of the DB Svc (DNS of the hosting Tenant Ring and the IP address of the host Node) is send back to the client. The client re-starts the connection process with the new information targeting the DB Svc node directly. When the connection reaches the host Node it is picked up by the Socket Duplicator component. The incoming connection is routed to the SQL Server instance that actually hosts the target database using TCP socket duplication.
  The redirection is a preferred mode of connecting, as it offers the lowest latency, once the connection is established.
  It requires however the newest client drivers that support the redirection in TDS protocol, and it requires

customers to open up more ports then just 1433. Therefore, this mode is not suitable for all customers.



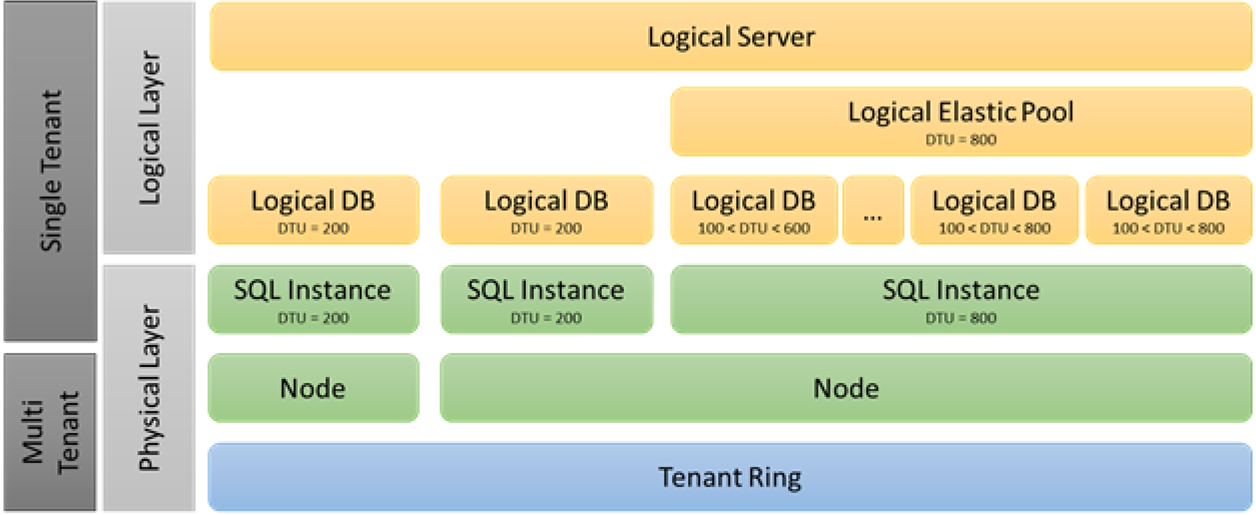
- **Proxy** – this mode is available as a backward compatibility option. When the connection is received by GW node, the same DB Svc lookup happens, however the connection is not returned to the client, instead a new connection to the backend DB node is created from the Control Ring and the two connections are used to proxy packets from the client over to the backend. The connection handling on the DB node is the same as described above. This mode requires the GW node and the proxy process to be available throughout the connection lifetime and it puts more load on the proxy code however it can support any clients.

Resource Governing

- **Resource Limits** – The implementation of Resource Governing relies on a combination of resource limits set through NT Job mechanism and internal SQL Server resource governing. For each user database or for a pool of databases we allocate a SQL Server instance (sqlservr.exe process) running as Service Fabric application and guarded in limits of an NT Job. The NT Job limits define CPU affinity and CPU rate caps as well as memory cap. IO limits are governed by internal SQL Server resource governing, implemented through SQL OS layer. Storage limits are set through NTFS quotas. In case of the pool, the per-databases limits are enforced through internal SQL Server resource governing inside the sqlservr.exe process.

Security

- **Isolation** – Each SQL Server physical instance can only host databases from one logical server and therefore from one tenant. Each instance under a low-privileged account, each only having write permissions to its persisted data and read on specific local executable files. Should the server process be compromised, the attacker can only access data of that instance. Wider attack would require using an additional vulnerability to get access to the OS level on the node.



Management

- **CMS** – Management operations are orchestrated through the Management Service component running in the Control Ring. The MS service listens on the incoming REST API requests and initiates workflows to handle them. The workflows are implemented through a combination of multiple Finite State Machines (FSMs) backed by persisted Cluster Metadata Store (CMS). The CMS is running on a Premium-level instance of SQL Server hosted in the Control Ring. The FMS code operated by an FSM Runtime hosted in the Management Service ensures that all operations are carried our reliably. The reliability is guaranteed by persisting each FSM transition in the CMS database and using SQL Server transaction semantics to ensure ACID properties of such transitions. If the state transition fails either because of external error, or because the Management Service carrying it has restarted, it will be retried until success or until another operation cancels it.
- **TDS Redirection** – Since some management operations, such as creating or deleting a database, can be carried out through T-SQL language over the TDS connection, we implement a redirection of such requests to the Management Service, to make sure we have a single logic and single place to carry such requests. When a T-SQL request is received in the SQL Server, it is parsed and instead of being executed locally it is

sent through an HTTP request to the Control Ring, where it follows the same workflow as if it were initiated through a user REST API call. The backend SQL Server instance authenticates to the Control Ring with a unique per-instance certificate to guarantee that it can only carry operation on the Logical Server to which it belongs.

Disaster Recovery

The Sterling service provides automatic Disaster Recovery capabilities, which allows users to restore the data that became unavailable either due to a data center outage (disaster) or due to a user-induced problem, such as accidental data deletion.

To support that, each backend DB Svc instance is running a Backup Manager process alongside the SQL Server instance process. The Backup Manager process is performing backup/restore operations on the instance.

- **Backups** are taken periodically, and a backup schedule is defined comprising of full, incremental, and log backups to allow us to provide required recoverability guarantees. Those backups are stored in a Azure Storage BLOB containers in the same region as the backed up server. In addition to that the backups are also copied to a secondary geo location which is remote from the original one. Also we create a remote replica of CMS database, holding all the metadata about the servers. The combination of data and metadata stored in the remote geo-location allows us to provide a restore service to restore a server even if the region where it originally was placed becomes completely unavailable. The restore operation can be initiated by the customer using our management REST APIs routed to the secondary backup location. The restore operation is completely controlled by the customer.

The local backups are used if customer needs to perform Point-in-Time Restore (PITR) to the same or a new server in the original server location. This is typically used to roll back some accidental changes to the database. This operation is also orchestrated through REST APIs.

Telemetry and Monitoring

The telemetry and monitoring in Sterling is build based on all the standard components available in Azure ecosystem – MDS, MDM, AIMS alerting, Cosmos and so on.

- All components emit events. We have standardized on usage of XEvents (SQL Server high-performance eventing mechanism) where possible, and on ETW for non-SQL components.
- The events are stored locally on the node first and then processed and uploaded by local MDS Agent over to the MDS store. The MDS Agent can also upload additional information, such as SQL Server error logs, node logs, performance counters and so on. Before uploading the data is scrubbed to ensure no PII information is transported to MDS.
- Once the data is in MDS it is processed through a “hot”, “warm”, and “cold” pipelines. The “hot” pipeline is used to drive the dial-tone alerts. It uses MDM to generate alerts. The “warm” pipeline provides regular alerts that include AIMS-based alerting and “Smart” alerts. The “cold” pipeline takes the raw MDS data and pushes it to Cosmos where it is processed and aggregated and allows us to compute system KPIs, observe the long-term trends and patterns. The aggregated data in Cosmos is stored for several months. The raw MDS data is kept for a week.
- In addition to simple AIMS rules we also use Machine Learning –based algorithms to detect changes in telemetry patterns and find anomalies. Those algorithms are trained to the typical workload pattern and alert on any changes from it.

- For troubleshooting of life-site issues we use a set of views hosted in an XTS troubleshooting tool and using MDS data. There are per-component views that allow engineers to quickly narrow down to the root cause of the problem without re-typing complex MDS queries directly in the MDS viewer. To speed up some of the troubleshooting we also utilize Kusto, which is a search index over MDS.

Number of components in Control Plane and in Data Plane use external monitors or runners to monitor the availability of externally-facing endpoints. The runners ensure that we detect endpoint unavailability, as well as generate steady load on the system to provide flow of telemetry to the alert rules.

We have also invested a lot in troubleshooting tools and services. To name a few key areas:

- XTS is our unified troubleshooting tool – engineers define troubleshooting views for their components which pull data from MDS or from Kusto and combine with cluster metadata coming from CMS database. The views are tailored to troubleshoot specific scenarios, such as provisioning, performance, HA, and so on. Every engineer on the team is familiar with the tool and how to navigate it and familiar with all key views required to understand the health of the cluster. We have a dedicated set of engineers continuously improving that tooling.
- Cluster Admin Service endpoint and cmdlets – we provide a set of PowerShell cmdlets to perform cluster management operations through a Cluster Admin Service endpoint hosted on Control Ring for each region. This interface allows us to operate in DevOps mode, where every engineer can mitigate customer issues and alter cluster state as required, with all the proper authentication and authorization checks and with auditing in place.
- Auto-mitigations through a BOT – we provide an automated BOT process that can react to alerts and perform common mitigations automatically thus saving the time of human engineers.

Deployment

- Each component in the Sterling service is a separate Service Fabric application and can be deployed independently of the other applications. There are no ordering or timing dependencies between applications and services. This allows them to be deployed independently without requiring serialization and coordination between components.
- Similarly, there is no specific ordering required between Control Ring and Tenant Ring deployments. If there are any component dependencies, the code is guarded by in-code Feature Switches, enabled through Service Fabric configuration updates. The Feature Switches are enabled only when all required components are deployed successfully. This is further guarded by a Dependency Framework that keeps track of which switch depends on what versions of what component and prevents accidental enabling of a feature.
- Tenant Rings deployments can proceed in parallel and do not impact availability of each other.
- The scalability of the deployment automation is such that most deployments can be completed in 4-6 hours allowing us to successfully carry most of the deployments during business hours.
- The alerting system has a built-in feedback mechanism to deployment allowing us to stop an in-progress deployment if it causes an incident to prevent wide-spread outage.
- Since application deployments are independent, each component team can schedule and orchestrate their own ones independently. There is a centralized system we use to track all in progress deployments (https://shinkansen)

How good have you found this content?

🙂 🙁