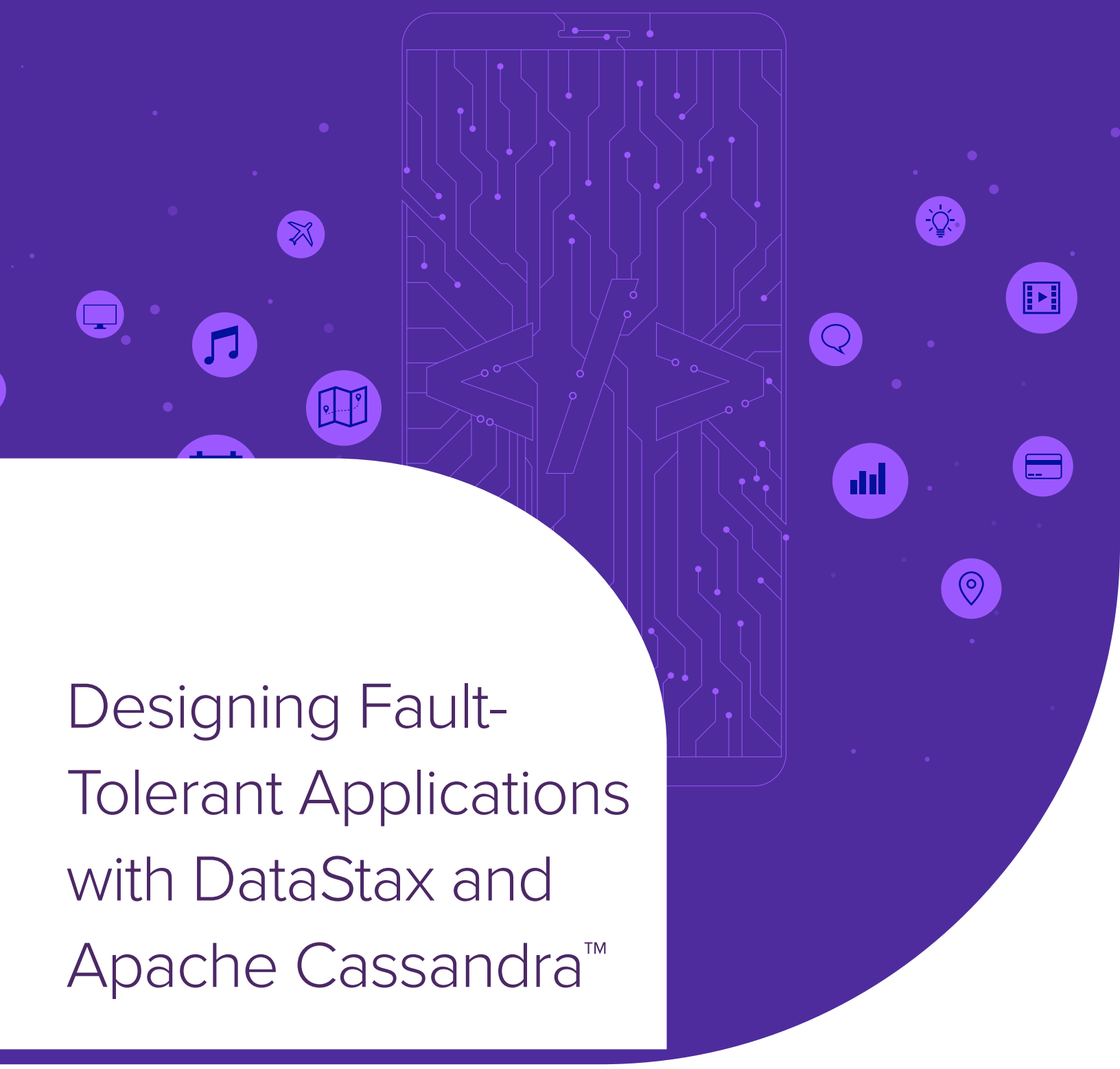


Designing Fault-Tolerant Applications with DataStax and Apache Cassandra™

DATASTAX®

CONTENTS

DESIGNING FAULT-TOLERANT APPLICATIONS WITH DATASTAX AND APACHE CASSANDRA™

DataStax's Distribution of Apache Cassandra, DataStax Enterprise (DSE), and Apache Cassandra are designed to be globally distributed and highly available with best-in-breed multi-data center replication and no single point of failure because of their masterless architecture. C&S Wholesale Grocers explains the value and impact of these database characteristics to their business:

"We needed an application that was entirely reliable and not vulnerable to unplanned outages because our warehouses are pretty much 24/7… the data integrity, reliability, availability, and speed were the main reasons we went with the DataStax solution."

—Salil Sinha, Vice President of IT Systems at C&S Wholesale Grocers

This powerful functionality enables greater flexibility when developing and deploying your cloud or on-premises application to deliver a fault-tolerant, always-on experience.

In general, there are various scopes of failure in a distributed environment that should be accounted for when designing resilient applications. Specifically, an individual server instance may falter, a cloud provider's availability zone (AZ) or local rack may experience issues, or in more drastic cases, an entire cloud provider's region or an on-prem local data center may be taken offline.

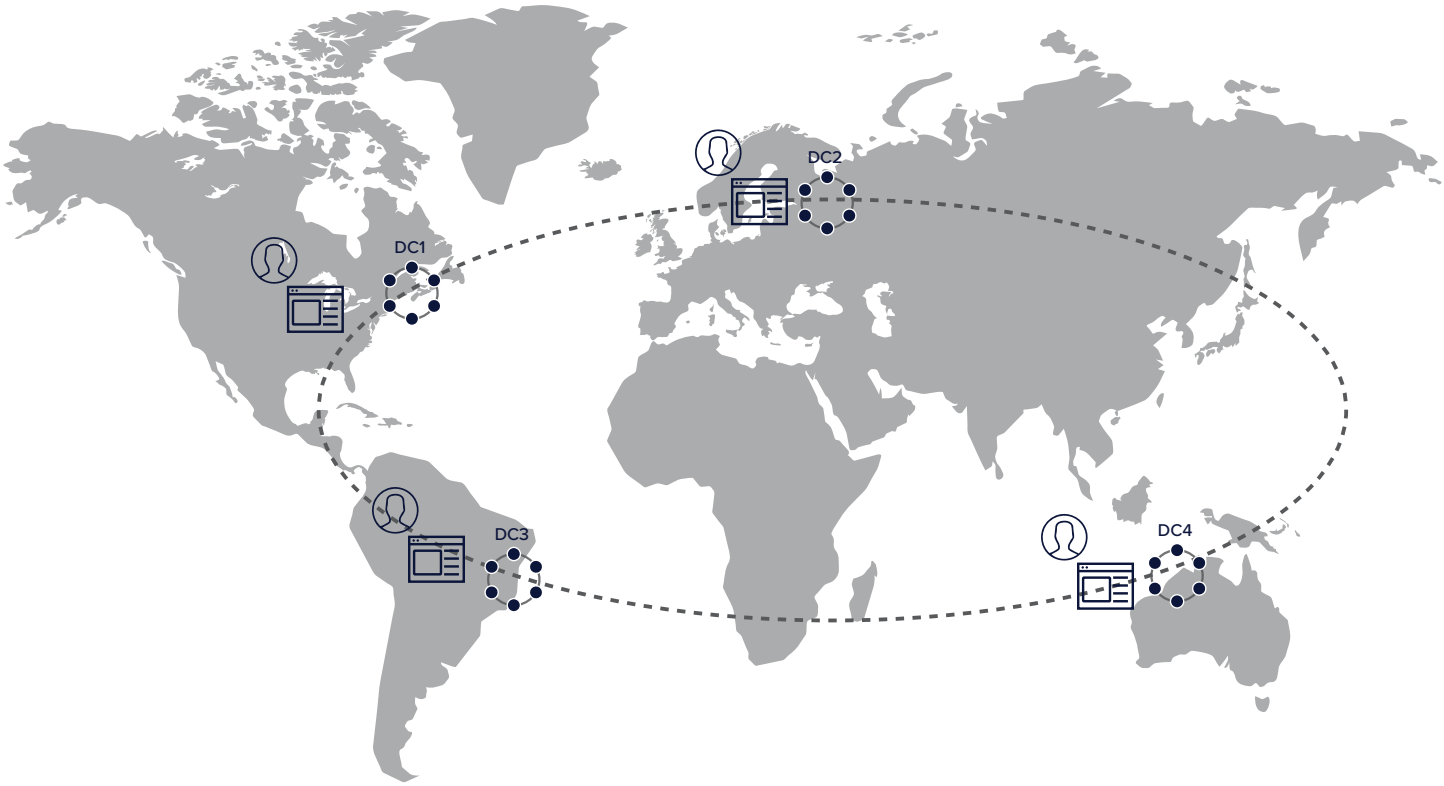
In the context of public cloud providers, an availability zone consists of one or more physical data centers, each with redundant power, networking, and connectivity, while a region is a physical location in the world that consists of multiple AZs. The equivalent of availability zones for on-prem deployments is a group of racks with redundant power and connectivity. The on-prem parallel to a cloud region is a physical data center location, composed of multiple groups of racks.

A single server instance may experience an array of issues including disk failure, networking or connectivity partitions, among other factors that could render the instance inoperable. Availability zone and region-wide outages are more unusual, but do occur. When such outages happen, recovery from the problem can take several hours to days and surely causes havoc for businesses across industries. Uptime Institute's 2018 Data Center Survey polled nearly 1,500 respondents and key findings revealed that nearly a third of all reported outages cost more than $250K; 41 respondents reported a single outage cost over $1M; and one specific incident cost over $50M. The risk is real and shines a light on the importance of designing a resilient architecture to prevent this from happening to your company.

How do DSE and Cassandra stand out from the pack in these scenarios? Read on to find out!

DATA CENTERS AND REPLICAS IN DSE AND CASSANDRA

DSE shares the same topological design as Cassandra, organizing a group of nodes into data centers and racks. Setting up the nodes into logical data centers in DSE allows you to geographically distribute your data and locate it physically close to your users, minimizing traffic between regions and reducing overall latency.



Users around the world access the DSE data center closest to their location.

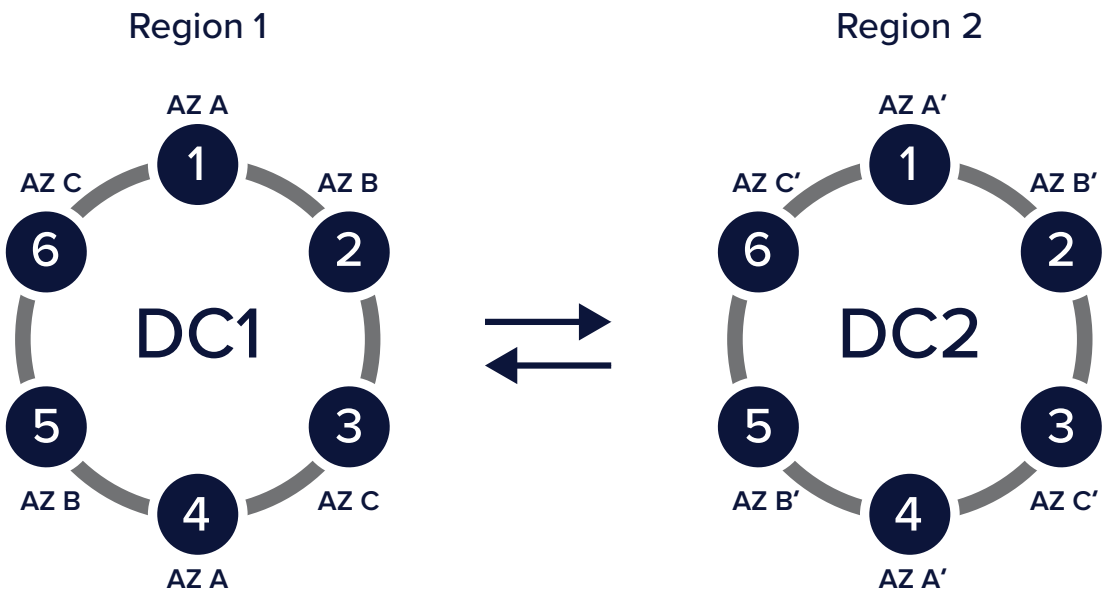
You can control how data is replicated across the different logical data centers when creating a keyspace in DSE or Cassandra. We refer to them as "logical data centers" because they may be placed in the same or different physical locations and the primary purpose is to tell the system how to replicate the data.

```
CREATE KEYSPACE orders
WITH REPLICATION = { 'class': 'NetworkTopologyStrategy', 'dc1': 3, 'dc2': 3 };
```

In the example above, data is replicated across three nodes in the logical data center named "dc1" and across three nodes in "dc2."

As discussed previously, availability zones (or racks) are a single failure domain within a region (or physical data center), and nodes should be deployed across different availability zones within each region to account for this failure scope.

Without going too deep into the weeds, DSE and Cassandra use tokens to control the placement of data across AZs (racks) and regions (logical data centers). To correctly set up a cluster with an even token distribution across each availability zone it is recommended to use the automatic built-in token allocation algorithm that is activated through the allocate_tokens_for_local_replication_factor setting in cassandra.yaml (note: this setting is in DataStax's Distribution of Apache Cassandra and DSE only, for Apache Cassandra see allocate_tokens_for_keyspace). Reference the documented procedure for initializing multiple nodes and data centers.



One data center per region, provisioning nodes across different AZs in order to be resilient to outages at this level. Masterless architecture of DSE will keep data in sync across the data centers.

CONSISTENCY LEVELS

In the context of DSE and Cassandra, the consistency level determines the number of replicas that need to acknowledge the read or write operation success to the coordinator of the query. To be clear, all replicas in the cluster will eventually receive a given write, the consistency level just controls the number of replicas that need to answer a given request for it to be considered complete from the client's view.

When designing an application, it is important to consider the number of replicas that must receive or serve the data based on your business requirements. As an example, if you are processing transactions and must always read the most recent write across regions, you will need to use stronger consistency levels. On the other hand, if you are storing ratings for a given product or are using Spark for a batch job, you probably won't have the same requirements, as it could be tolerable to have "old" reads. Therefore, you could use looser, more eventual consistency levels.

For the purpose of this paper, we'll focus on strong consistency levels that tolerate some degree of failure in cluster deployments with multiple logical data centers. Those consistency levels are defined below.

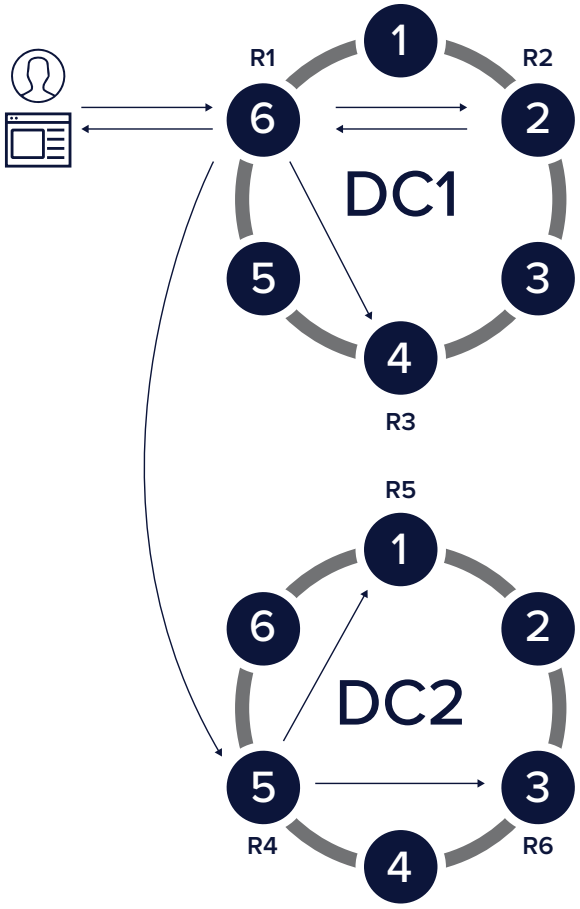
- LOCAL_QUORUM: A majority of the replicas in the local data center must respond. The data center here is a group of nodes that were configured to belong to the same logical data center in DSE/Cassandra. That data center is considered local by the driver when its name is passed to the driver load balancing policy as the local data center. In this case the driver will only create connection pools to the nodes in that data center and will use that group of nodes as coordinators for requests.
- EACH_QUORUM: A majority of the replicas in each data center must respond.

Writes performed at LOCAL_QUORUM have to be processed by a majority of replicas in the local data center only. So in the above example, where we have three replicas in "dc1" and three replicas in "dc2," if "dc1" is our local data center, then two of the replicas in "dc1" must acknowledge the request for it to be considered successful by the client.

Writes performed at EACH_QUORUM have to be processed by the majority of replicas in each of the logical data centers. So in the above example, where we have three replicas in “dc1” and three replicas in “dc2,” two of the replicas from “dc1” and two of the replicas from “dc2” must acknowledge the request for it to be considered successful by the client.

In both cases, the coordinator of the query will send the write request to replicas in all data centers, the difference relies on whether the coordinator should wait for remote replicas to acknowledge the write for the operation to succeed.

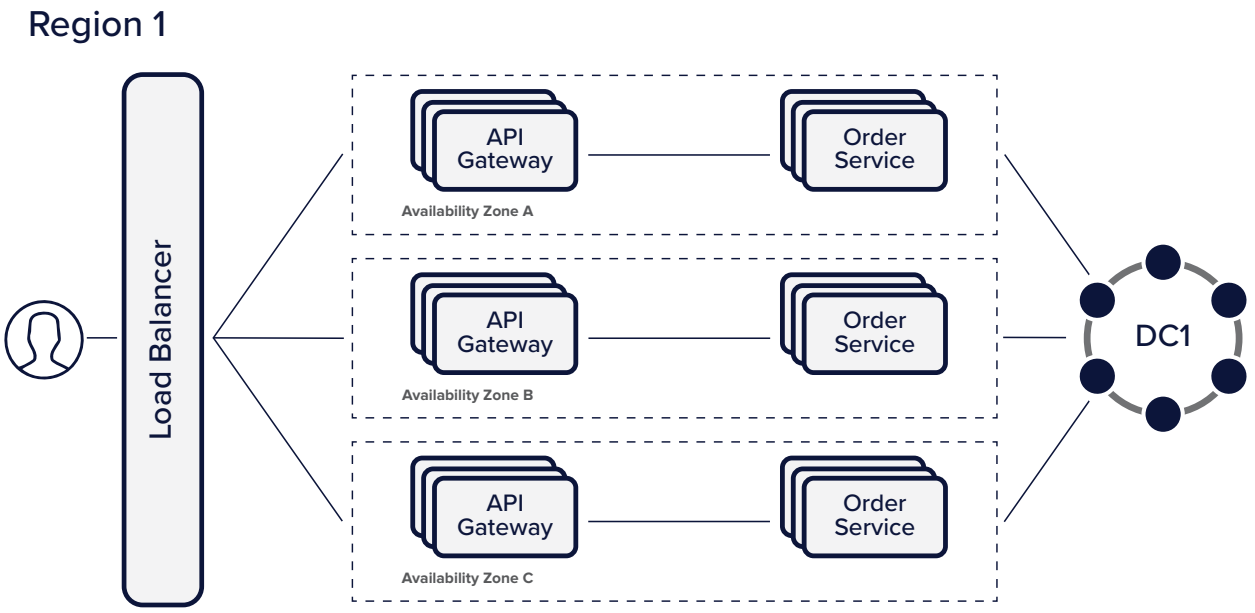
In DSE and Cassandra’s tunable consistency model, choosing the consistency level is a trade-off between availability and data consistency. EACH_QUORUM writes guarantee that the majority of replicas in each data center received the write but requires more nodes to be available and to confirm the write, whereas LOCAL_QUORUM writes guarantee that the majority of replicas in the local data center received the write and only requires those local nodes to be available and to confirm the write for a request to succeed. Note that there is also a performance trade-off with consistency levels, such that queries that require a larger number of nodes to acknowledge the operation will almost always take longer than queries that require acknowledgement from a lesser number of nodes, especially for geographically distributed architectures.



The coordinator of the request only waits for acknowledgement from two of the three local replicas. In the example above, the coordinator is a replica which is the case when using the default token-aware routing.

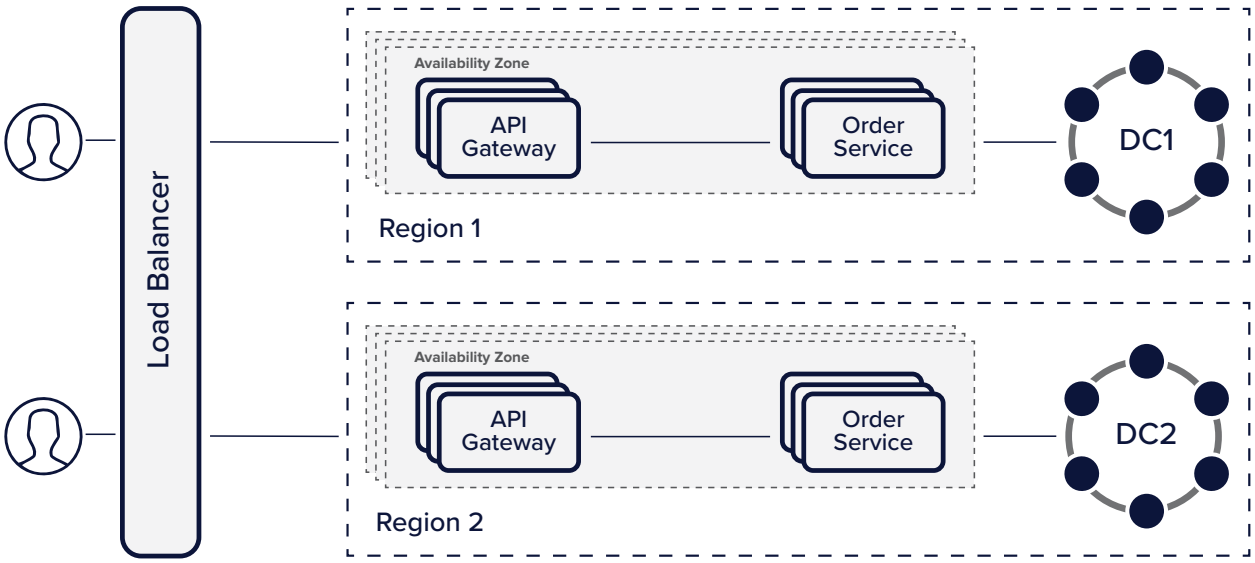
APPLICATION ARCHITECTURE

Now let’s look at how we could design a globally distributed application architecture that is resilient to the different infrastructure failure domains. The following sample diagram shows a microservice application architecture for a single region.



The client in this diagram hits the load balancer (LB) after the domain name system (DNS) resolves the name of the host. The LB will then distribute the traffic within the region to the API gateway service instances in an availability zone. The API gateway then routes the traffic to each microservice instance, that in turn sends the database requests to the nodes in the local data center that the DataStax driver is connected to. For simplicity of the diagram, we show only a single service type "Order Service," but the same principles apply to applications that are composed of many services.

We expand this pattern of resource isolation for multiple regions in the following figure.



After initial DNS resolution, a load balancer instance routes traffic within the region.

Note that a single load balancer for all the regions is shown in the diagram for simplification.

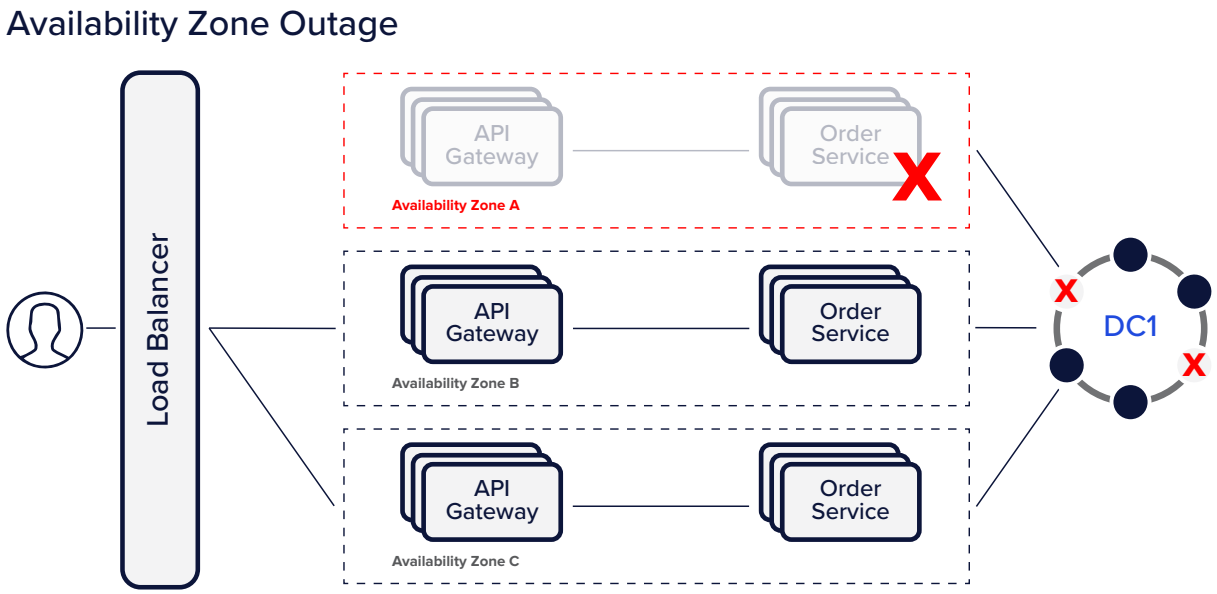
Global load balancing can be achieved in two different ways:

- A DNS service that resolves to a single anycast IP address, where the IP address front-ends instances of the load balancer in regions around the world. The anycast mechanism will select the region that is geographically closest to the client. In the case of a region-level outage, the load balancer automatically moves traffic to other available regions. In the public cloud space, Google Cloud Platform (GCP) Load Balancer, Amazon's AWS Global Accelerator, and Microsoft Azure Front Door Service feature this type of routing.
- A Geo DNS service that resolves to the IP address of the load balancer instance in the region closest to the client. The load balancer then distributes traffic within the same region. In the public cloud space, AWS Route 53, GCP Cloud DNS, and Azure Traffic Manager act as geo DNS, and there are tools for managing DNS across multiple providers like octoDNS.

In a nutshell, the difference relies on whether the global traffic is funneled to the regions by the load balancer or by DNS resolution.

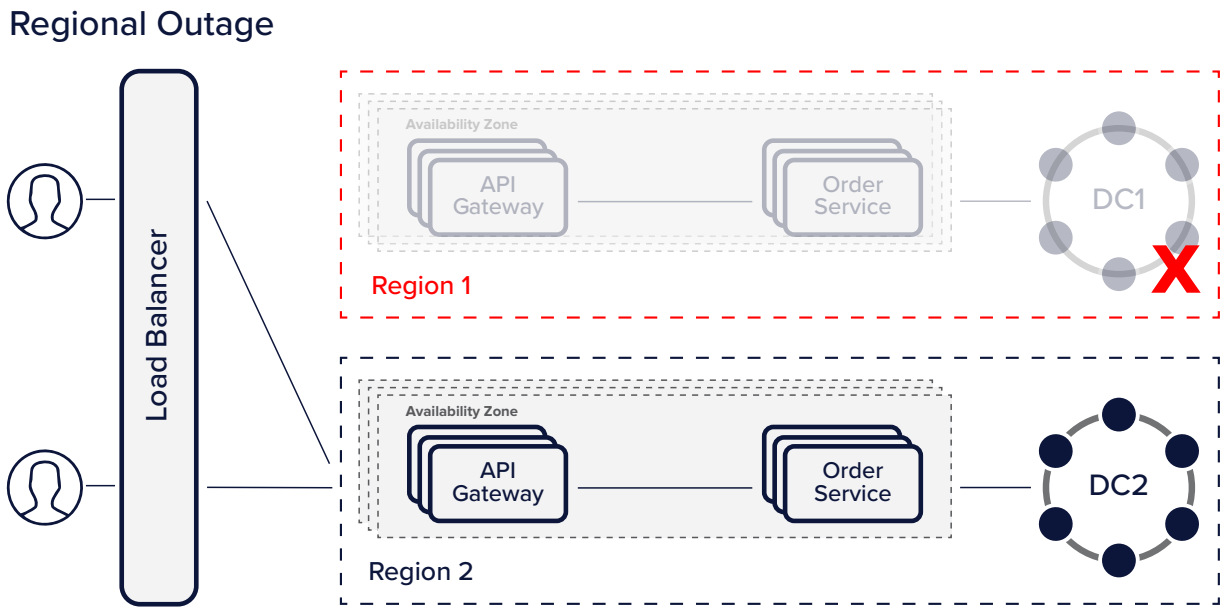
OUTAGE SCENARIOS

Let's try to illustrate outages at different failure domains.



In the case of an availability zone outage, the load balancer will route traffic to API gateway instances in the healthy zones.

At the database level, DSE and Cassandra are designed to be resilient to replica failures and continue to operate normally when provisioned correctly. Application service instances using DSE from availability zones that were not impacted will not experience errors derived from the loss of connectivity to the failed nodes, and the DataStax Drivers will attempt to reconnect in the background while using the set of live nodes as coordinators for the queries while the failed nodes are offline. There are also mechanisms in the driver to proactively send requests to additional nodes in order to avoid application-side impact from individual node failure.



For region-level outages, the initial data flow will depend on the type of global load-balancing mechanism used:

- With anycast-based load balancing, the traffic from clients geographically close to the unavailable region is distributed to the healthy regions automatically.
- With Geo DNS and regional load balancer instances, the client will have to re-resolve the host name to obtain the address of one of the healthy regions. This could take some time, in particular the time to live (TTL) of the DNS record plus the healthy check intervals and thresholds.

It's worth noting that, in both zonal and regional outages, remediation at the service level is not required thanks to the distributed nature of DSE. Application services that are healthy can continue querying the database targeting the local data center.

Previous versions of the DataStax Drivers featured selecting nodes from remote data centers as coordinators when the local data center was detected down when using the data center aware load balancing policy. In contrived scenarios, switching from the application layer to another healthy data center seemed like a good idea. After seeing the same support tickets coming in time and time again and analyzing the situation in greater detail, we decided to deprecate and eventually remove these driver features as it became clear that this approach has several pitfalls:

- If there's something wrong in your application or data model that brought the local data center down, you risk propagating this issue to the rest of the cluster, ultimately bringing the whole thing down.
- In the case of a geographical infrastructure outage, failing over at the driver / application level won't help you, as your application services will also be unavailable.
- Sporadically failing over to a remote data center will likely cause latency spikes as load transitions to a non-local data center.
- When using LOCAL consistency levels, guarantees of reading your own writes can be lost unexpectedly and non-deterministically.

- Cross-region or inter-data center traffic might have additional costs, and boundaries should be carefully put in place to account for bandwidth and costs.
- If you are using DSE Advanced Workloads, having your queries fail over to a remote data center may cause queries to fail if the remote data center does not have the same workloads enabled as the local data center.

In reality, this type of geographical failover should be architected at a higher layer in the stack. It is critical to think about this problem holistically and to address these failure scenarios across each layer of your system. Specifically, the impact of these types of failures should be anticipated and thought should be given to proper provisioning and auto-scaling as well as defining application data guarantees during these events.

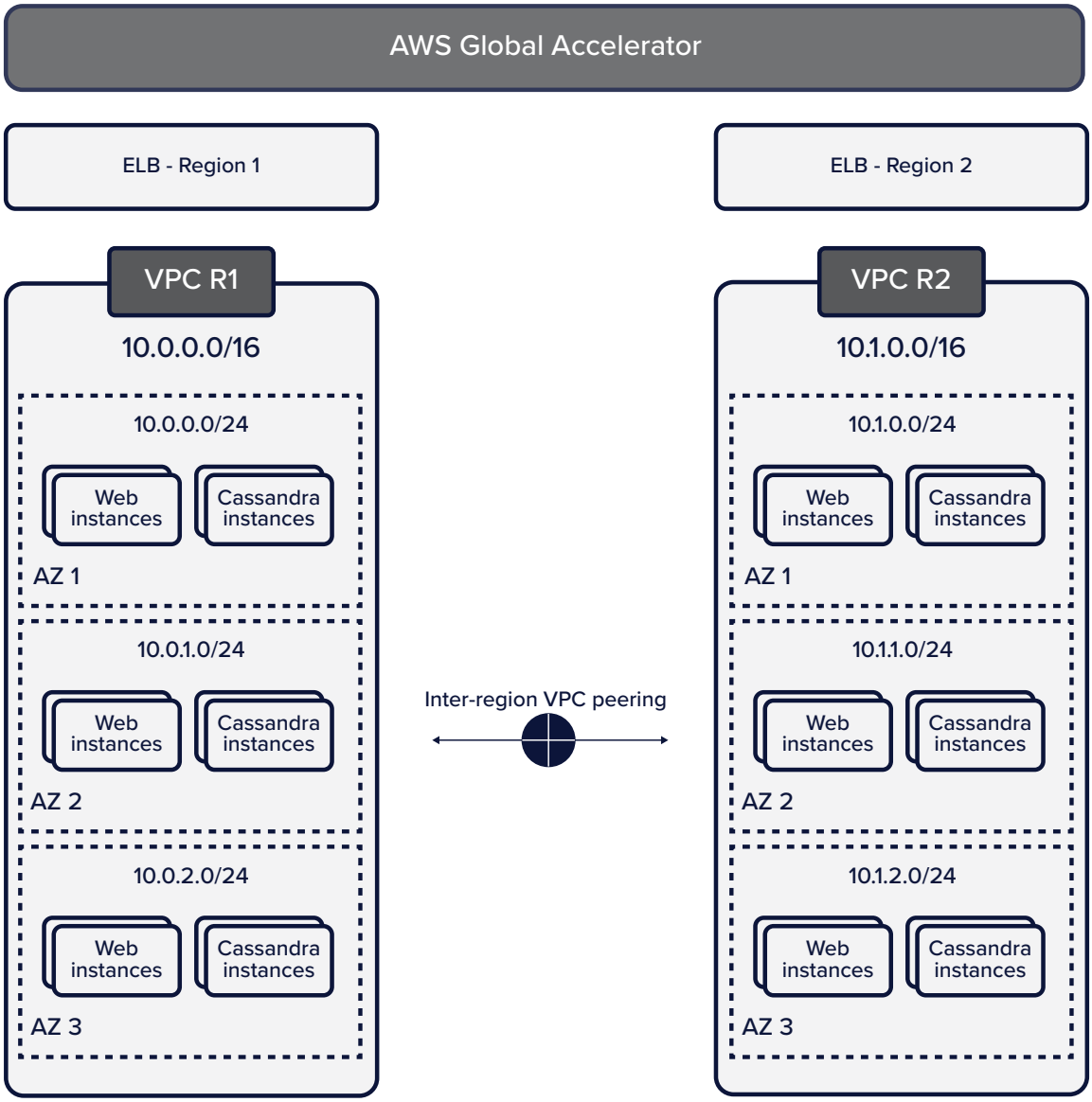
PROVISIONING / SCALING

In order to tolerate infrastructure failures gracefully, you need to design your system in a way that healthy components are able to handle the additional load derived from the loss of parts of your infrastructure.

At the application service level, auto-scaling can be achieved by creating stateless microservices that can be added without disrupting the existing application. For the database portion of the stack, in order to maintain stability in the case of availability zone or region failure, it is necessary to have adequate capacity on the healthy machines to accept the additional load of traffic from the failed machines during this type of event.

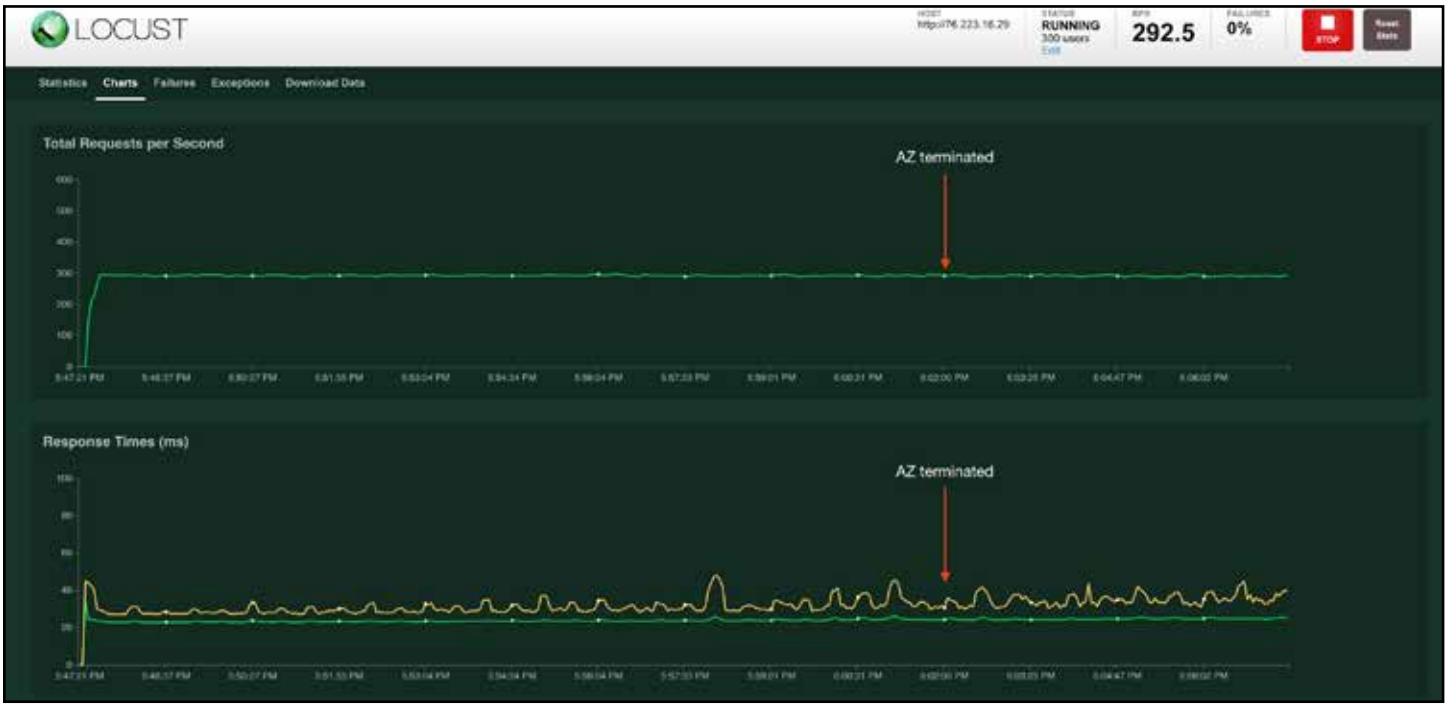
CONCLUSION

We know that delivering a truly resilient application is easier said than done, so we've created provided a working demo that implements exactly what we advise in this white paper. We hope that this will accelerate your implementation in your own technology stack and show the true value of a masterless, geographically distributed database that can be deployed anywhere, on-prem or in any cloud.



Failover experiment architecture

In this demo, we simulated availability zone and region failure while pushing requests to the AWS Global Accelerator via Locust clients in each region. We intentionally designed this demo to highlight availability rather than raw performance and application / data model thoroughness. Below is a screenshot of the client while simulating an availability zone infrastructure outage.



Availability Zone was terminated at 6:02 PM, note the lack of impact on throughput and latency.

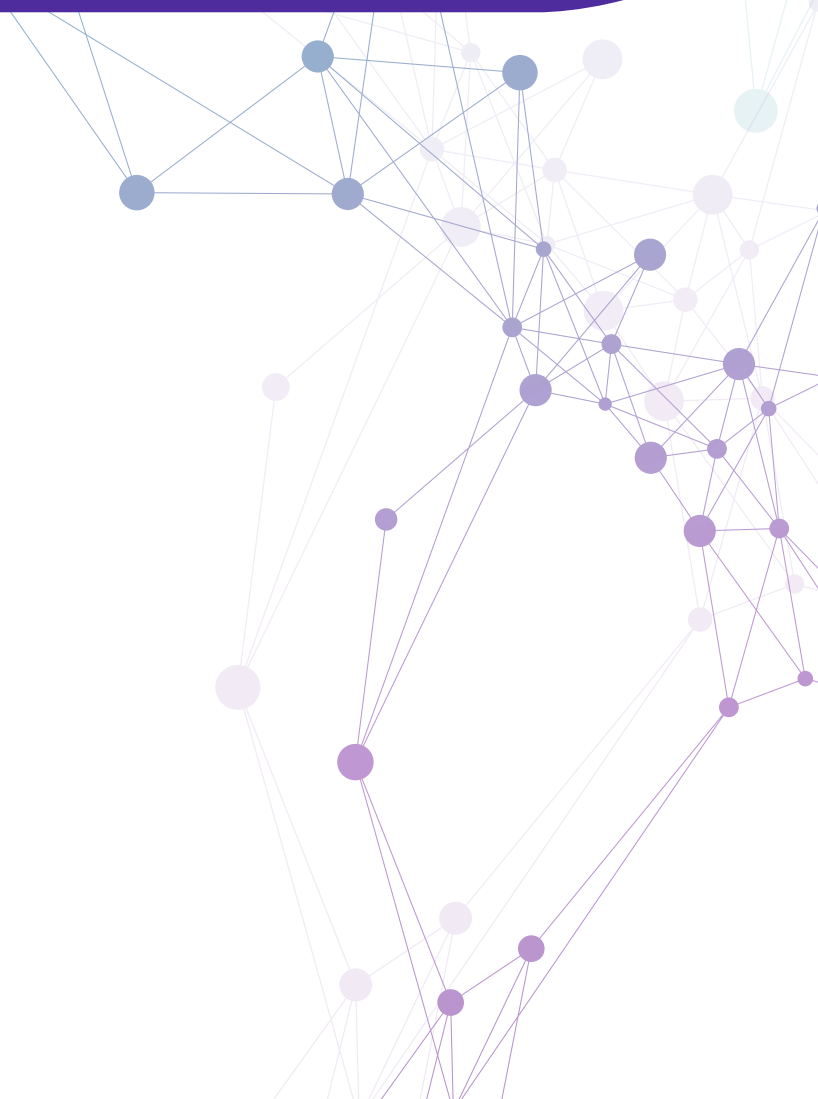
Save yourself from those heart-dropping calls in the middle of the night and execute a bulletproof architecture using DataStax as the database in your stack.

Further Reading

- DataStax Architecture Guide
- DataStax Developer Guide
- Amazon Route 53, DNS Failover
- Netflix Tech Blog – Active-Active Multi-Regional Resiliency
- Netflix Tech Blog – Flux: A New Approach to System Intuition
- *Netflix Tech Blog – Global Cloud—Active-Active and Beyond*
- *Analysis: Rethinking cloud architecture after the outage of Amazon Web Services*
- *AWS Reliability Pillar*

ABOUT DATASTAX

DataStax delivers the only active everywhere hybrid cloud database built on Apache Cassandra™: DataStax Enterprise and DataStax Distribution of Apache Cassandra, a production-certified, 100% open source compatible distribution of Cassandra with expert support. The foundation for contextual, always-on, real-time, distributed applications at scale, DataStax makes it easy for enterprises to seamlessly build and deploy modern applications in hybrid cloud. DataStax also offers DataStax Managed Services, a fully managed, white-glove service with guaranteed uptime, end-to-end security, and 24x7x365 lights-out management provided by experts at handling enterprise applications at cloud scale. More than 400 of the world's leading brands like Capital One, Cisco, Comcast, Delta Airlines, eBay, Macy's, McDonald's, Safeway, Sony, and Walmart use DataStax to build modern applications that can work across any cloud. For more information, visit www.DataStax.com and follow us on Twitter @DataStax.



Last Rev: AUGUST 2019