Edit this page

Cell architecture

Learn why cells are used, their basic architecture, which team owns each component, where to go for migration help, and find links to additional resources for understanding how to interact with cells.

Overview

This section of the New Relic Platform docs provides relevant resources to understand cell architecture and understand the tools available for migrating services from the CHI and EU datacenters to cells.

Cell Architecture is our technical strategy to:

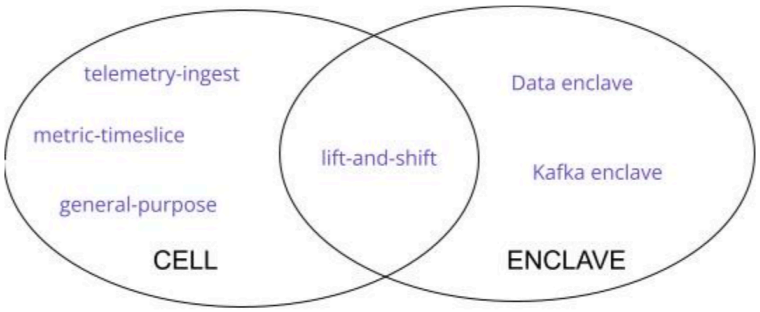
- Enable an incremental move to the public cloud
- Unlock repeatable scaling for our services
- Limit the blast radius of incidents

For more background info, discover our founding vision for the cloud platform

There are a few options for running services or finding shared resources in the cloud:

- **Cell**: ephemeral, infinitely replicable, runs your code, apps store data in enclaves
- **Enclave**: stateful, limited population, runs shared infrastructure
- **Lift and Shift Enclave (LSE)**: apps store data in enclaves, only one LSE at a time, runs your code

The image below helps visualize the relationship:



Cell types

Cells have different types that indicate their use case. To see the various cells and types, run `i9d cells list`. Many cells are infrastructure cells, but most run production applications. Infrastructure cells include things like routing cells.

GP cells

To avoid proliferation of special cases, most of New Relic's services should exist in what's currently called a General Purpose Cell (GPC or GP cell). Older docs refer to this as a General Pool cell). GP cells are stateless, ephemeral, infinitely replicable k8s environments which run your code.

- **Stateless**: GP cells run stateless services that don't keep any of their own state in their environment. Stateless services which need databases or Kafka topics rely on external services in enclaves (see below). This includes many of New Relic's microservices.
- **Ephemeral**: We originally envisioned destroying and rebuilding every cell every 90 days. We're not there yet, but that's what we're working towards. Cells can and will go away. TDP ensures that taking down a single cell doesn't completely offline your service.
- **Infinitely replicable**: GP cells are all identical. One way to scale up is to deploy more copies of a GP cell. If your service can't handle an arbitrary number of copies of itself, it is not suitable for GP cells.
- **Your code**: Deploy code to cells using Grand Central similar to how you deploy it to the data centers.

For additional details on General Purpose Cells, see Understanding GP Cells. Migrating to GP Cells? See Plan a migration to General Purpose Cells.

Enclaves

Cells are stateless, ephemeral, and infinitely replicable, but this model doesn't make sense for everything. Once we realized this, it led to the development of enclaves. Enclaves are stateful, long-lived, and unique. Enclaves don't run your code, and are only for shared services. We expect that most enclaves will be for shared services, rather than a target you can deploy to.

There are several common stateful resources: databases, redis, kafka, and zookeeper.

There are a few use cases for enclaves:

- Data enclaves run Redis, Postgres, and MySQL for common use. See the DB self-service guide.
- Kafka enclaves hold a Kafka cluster for common use.

See Kynapses also, which is a granular way to replicate data between clusters.

> We know we need a shared ZooKeeper in the cloud and are looking at the best way to support it.

Lift and Shift Enclave

What if your code needs to be unique? What if you need exactly one copy of your service? (The canonical examples here are an emitter or something like a stateful slackbot). What if you need more than one copy of your service but can't handle an arbitrary number? Maybe you have some intra-service consensus protocol that needs an odd number of running instances, and it has to be less than, say, ten.

Ultimately these apps should be made compatible with GP cells. However, we realize that not every app can handle GP cells, and we didn't want to force a lot of rewrites on a tight timeline. This led to the development of a Lift and Shift Enclave (LSE).

An LSE is basically a special instance of a GP cell. It shares some properties with a GPC: it is stateless and runs your code. It also shares some properties with an enclave: it is long-lived and unique.

> An LSE is temporary. You can move your code into it, but you have to have a plan to get out at some point. We haven't announced an official support horizon.

LSEs aren't as reliable as GPC. If there's only one LSE, it will sometimes go away. LSE doesn't scale as well, as there are inherent limits to the size of any k8s cluster. And there's more you need to manage. If you need to run another copy of your application, you're on your own. But they're a valid bridge between how DC/OS works and how GP cells work.

For more details on the LSEs, see our migration guide.

Custom cells

These are specialized cells that perform a specific function where the components need to be tightly coupled. Examples include Telemetry Ingest (TI) and Metric Time Slice (MTS) cells. See the existing custom cell types, or learn how to create a new custom cell type.

If you don't see a cell type that works for your service, reach out to #help-cumulus-migration. We'll work together to determine if you can rework your application slightly to fit in GPC or an existing cell type before deciding if a new cell type is warranted for your service.

Cell architecture basics

A cell is a unit of scale and isolation containing all relevant components for New Relic's product(s) to work. In turn, each cell is contained within one of the existing NR instances: `us`, `eu` or `staging`.

Cells run in AWS (and Azure soon) where the level of isolation is one AWS account per cell. Each cell contains AWS managed services (MSK, RDS, ElastiCache, etc) and at least one Kubernetes cluster. Difference between CHI and cell components are found in the cell migration guide.

> Although cells are currently based in AWS, work is already underway to consider a multi-cloud strategy using other platforms like Azure.

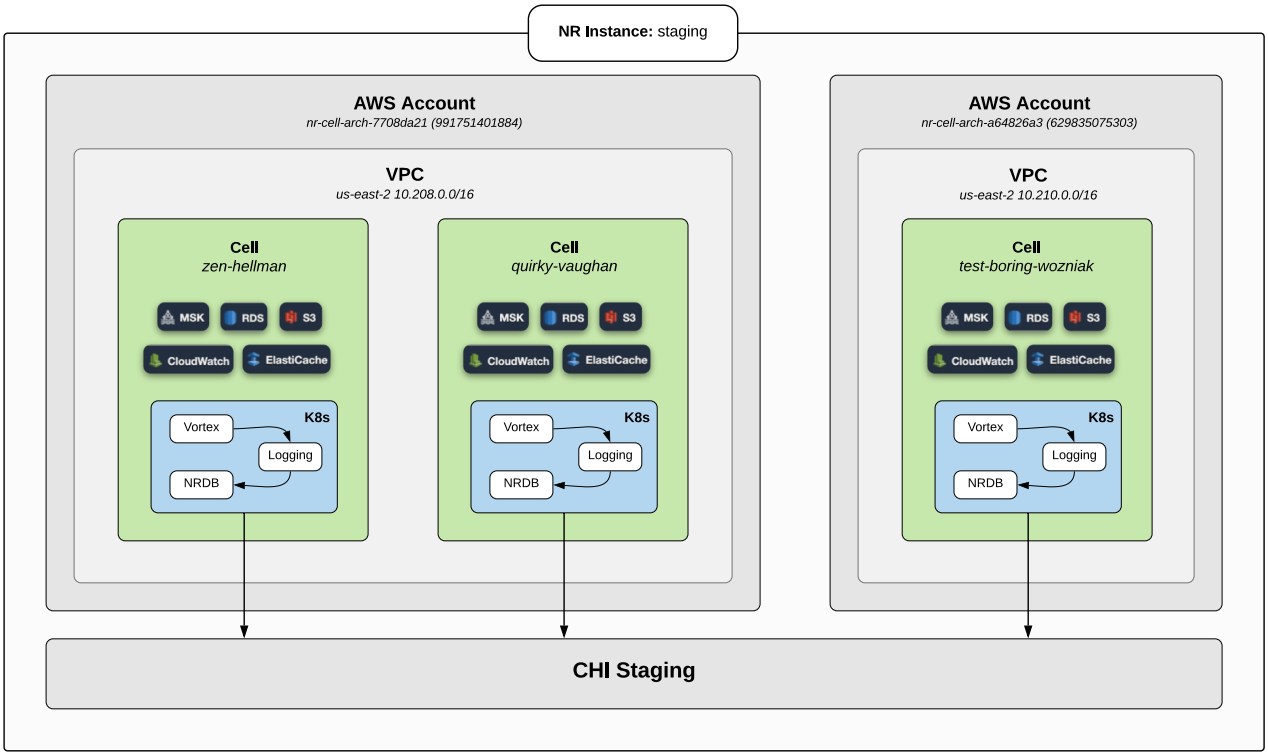
Cells are ephemeral and are regularly scheduled for build or decommission via Kitten Mittens.

See this presentation that provides an overview of cell architecture. Passcode: n+0$ujgz

The K8s cluster nodes are organized into pools:

- One general pool for most of the services
- Multiple dedicated pools for specialised workloads such as Vortex and NRDB

The diagram below shows how cells are laid out in the current `staging` NR instance:



Teams deploying to cells are responsible for setting the alerts for the services they manage and for being on-call.
Developers can use kubectl to inspect and troubleshoot their services.

Cell naming convention

Cells are given a globally-unique name comprised of the New Relic instance, an adjective, and a noun. Taking `us-sweet-edison` as an example, the New Relic instance the cell is a part can either be: `stg` (Staging), `test`, `us`, or `eu`.

These names are human- and machine-readable. They don't convey all possible cell details (like what workloads it might run), but they help humans in normal operations and incidents because they are short names limited to no more than 18 characters.

> Because of their ephemeral nature, cells are not renamed. Instead, a new cell is created and the older cell is decommissioned.

Migrating services to cells

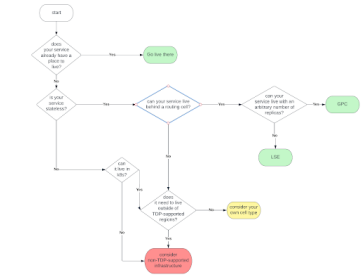
Teams migrate their services from the Chicago datacenter to cells by choosing one of the following options:

- Move stateless services to General Purpose Cells (GPC)
- Move to the Lift-and-Shift Enclave (LSE)
- Use an existing custom cell or create a new one

Choosing where to migrate your service

For most services, this has already been decided. The first thing to do is check you team's Jira under the Cumulus roadmap. The dates and estimates may be wrong because they predate the Cumulus pause. Feel free to update them!

If the plan in Jira looks solid then you should stick with it. If you're still not sure where to go, or if you think the plan in Jira needs adjusting, use this flowchart:



[editorial note: this flowchart is a work in progress]

> Bias your decision towards running in General Purpose cells unless you can justify not doing so.

There are a few things that we need to still sort out to facilitate migration:

- FIPS-compliant crypto libraries - not all languages have one, and this impacts FEDRamp support
- External fixed IPs
- Custom-rolled VIPs and router setups (there are like 15 services that need special treatment)
- Zookeeper for shared services

If you have concerns, questions, or need migration help, contact the #help-cumulus-migration Slack channel. Also, review the following resources before discussing migration paths or plans with an architect:

- Understand what cells are and how they differ from CHI/Marathon deploys
- Access kubernetes clusters in cells and understand cell deployments

Getting help with Cumulus migrations

Cumulus has two main initiatives:

- exit the data centers
- exit the AWS Marathon cluster

Any work that fits into one of these initiatives is part of Cumulus. However, some teams have their own cloud computing environments outside of what the Platform, Data, and Intelligence (PDI) group offers. Those teams are also trying to harness PDI's overall cloud computing strategy. It's a bit of a gray area, but the tooling those teams must use resides in Cumulus and the people who support Cumulus also support this tooling.

However, strictly speaking, services already in AWS on your own infrastructure are outside the scope of the Cumulus project and don't have the same priority to the Cumulus team. So, if you have questions or problems with specific pieces of technology, first visit the support channels for those technologies instead of #help-cumulus-migration:

- #db-team Slack for database concerns
- #help-kafka-platform Slack for Kafka concerns
- #help-elb Slack for ELB concerns
- #cell-arch Slack for questions and collaboration about cell architecture

If you have questions around overall cloud migration or don't know where to start, first talk to your architect. If you don't have one (or you are the architect) feel free to ask in #help-cumulus-migration but please tag it :no-party-cloud: This work is lower priority than Cumulus-specific work.

> The Cumulus team is not capable of handling last-minute, time-sensitive, spicy problems where you're unsure where to start. Don't wait until the last minute to determine what you need help with.

If you need help getting started with your migration from your cloud infrastructure to PDI's, here are some other general guidelines:

- PDI provides a k8s environment to run your containers, and has three vehicles for doing so: GPC, LSE, and your own cell type. We also provide a data enclave and a Kafka enclave for your DB and Kafka workloads.
- if a container can be replicated at will, it fits in GPC
- if a container needs replication control, it "temporarily" goes to LSE and then we plan where it will live long-term
- if it goes in GPC or LSE, its databases go in the DB enclave if, and only if, what you're doing is supported by the DB team. Data enclave is not for one-offs and special stuff
- if you need EC2 instances, they shouldn't live in the data enclave
- if a containerized service has very specific needs at scale, maybe it gets its own cell type and we'd have to have some more discussions to decide anything bare metal or outside the container realm is outside of the Cumulus scope. Take these matters to your architect or perhaps to #architecture but not a help thread like #help-cumulus-migration
- if you're running your own VPC, connect to the TGW in whatever region(s) you're in, but that's a #help-neteng thing, not Cumulus

Compliance in the cloud

Compliance is a tricky thing in the cloud. If you want to learn more about it, read our compliance documentation. There's more coming to make it easier to understand compliance requirements and this section will be updated with with those details.

Cell architecture component ownership

Check Team Store for email and slack channel details for the following teams or see support contacts.

Domain	Team
Architecture	Cell Leadership team (above)
Argo	Tools team
AWS ELBs in cells (load balancers)	Container Fabric team
AWS account access	Cloud Secrets & Identity
AWS resource creation (e.g. KMS, S3, ...)	requesting team
AWS tagging	requesting team
Account-based routing	Production Engineering
Cell builds and decommissions	Production Engineering
Cloudflare vendor relationship	Ingest
Direct Connect from cells to Chicago	NetEng team
Grand Central Kubernetes deploys	Tools team
Kafka mirror consultation	Kafka Platform Team
Kafka	Kafka Platform Team
Kubernetes clusters	Container Fabric team
Logging product	Logging team
Metrics product	Unified Data Streams team
NRDB operator	NRDB SRE team
NRDB	NRDB teams
Relational DBs	DB team
Transit Gateway	NetEng team

Domain	Team
Zookeeper	Kafka Platform Team

Related resources

- Migrating Kafka applications to the cloud
- Cumulus program page
- Cell build and decommission calendar
- Migrating to cells
- Accessing Kubernetes clusters

Page last modified: Jun 15 2022 at 08:00 AM.