Cloud Native Computing

Code Like Netflix

CloudNative.directory

Cloud Native Computing - The Future of Enterprise Technology

In a 2015 VentureBeat article the author explained how Netflix had pioneered the future of enterprise tech, an approach that came to be defined as 'Cloud Native'.

The practice is one of moving away from the traditional approach of owning and operating your own data centre populated by the likes of EMC, Oracle and VMware, and instead move to 'web scale IT', applications built with services packaged in containers, deployed as microservices and managed on elastic infrastructure through agile DevOps processes and continuous delivery workflows.

Going Cloud Native Boosts Share Price

In this day and age you wouldn't think having a web site would be a strategic factor.

However as the Independent reports the primary reason for the soaring share price of Dunelm was their web site upgrade, defining ***Cloud Native*** to be the essential success factor: *"We now have a modern, flexible, cloud-native platform that will be used to accelerate the development of our customer proposition," the group said.*

Similarly Air Malta attributes their turn around to the same transformation.

A recent ZDNet article lamented that CIO's are still struggling to define and achieve the use of technology as a tool for enabling competitive advantage, with the usual impediment of maintaining the existing legacy estate being the primary inhibitor.

'Going Cloud Native' represents a wholesale addressing of this challenge, a process of transforming that legacy estate to be entirely built for and operating in the Cloud, and with that, delivering the type of always on experience that customers expect in this digital age.

Code Like Netflix

With the approach now mature best practices are now well defined and readily accessible to organizations seeking the same transformation.

Indeed not only do they share their best practices via blogs, they also share the software they've created to make it possible via open source – Netflix OSS. You too can Build Code Like Netflix, literally. You can follow their updates on Twitter here.

The first major step for Netflix to become an entirely Cloud Native business was to migrate their on-premise legacy infrastructure, a journey begun in 2008 and completed in 2016.

They too previously operated a traditional enterprise data centre technology platform, where famously *"a single missing semicolon brought down the entire (monolithic) Netflix website for several hours in 2008."* Read more about this story on DZone.

Adrian Cockcroft, the chief architect behind this digital transformation, shares his experiences of the Netflix journey to the Cloud in this video.

In this blog they focus on the migration of the core Netflix billing systems from their own data centre to AWS, and from Oracle to a Cassandra / MySQL combination, emphasizing in particular the scale and complexity of this database migration part of the Cloud Migration journey.

This initial quote from the Netflix blog sets the scene accordingly:

*"On January 4, 2016, right before Netflix expanded itself into 130 new countries, Netflix Billing infrastructure became 100% **AWS cloud-native**."*

They also reference a previous blog also describing this overall AWS journey, again quickly making the most incisive point – this time describing the primary inflection point in CIO decision making that this shift represents, a move to '**Web Scale IT**':

"That is when we realized that we had to move away from vertically scaled single points of failure, like relational databases in our datacenter, towards highly reliable, horizontally scalable, distributed systems in the cloud."

Migrating Mission-critical Systems

They then go on to explain their experiences of a complex migration of highly sensitive, operational customer systems from their own data centre to AWS.

As you might imagine the core customer billing systems are the backbone of a digital delivery business like Netflix, handling everything from billing transactions through reporting feeds for SOX compliance, and face a 'change the tyre while the car is still moving' challenge of keeping front-facing systems available and consistent to ensure unbroken service for a globally expanding audience, while conducting a background process of migrating terabytes of data from on-site enterprise databases into the AWS service.

- *We had billions of rows of data, constantly changing and composed of all the historical data since Netflix's inception in 1997. It was growing every single minute in our large shared database on Oracle. To move all this data over to AWS, we needed to first transport and synchronize the data in real time, into a double digit Terabyte RDBMS in cloud.*
- *Being a SOX system added another layer of complexity, since all the migration and tooling needed to adhere to our SOX processes.*
- *Netflix was launching in many new countries and marching towards being global soon.*
- *Billing migration needed to happen without adversely impacting other teams that were busy with their own migration and global launch milestones."*

The scope of data migration and the real-time requirements highlight the challenging nature of Cloud Migrations, and how it goes far beyond a simple lift and shift of an application from one operating environment to another.

Database Modernization

The backbone of the challenge was how much code and data was interacting with Oracle, and so their goal was to 'disintegrate' that dependency into a services based architecture.

"Moving a database needs its own strategic planning:

Database movement needs to be planned out while keeping the end goal in sight, or else it can go very wrong. There are many decisions to be made, from storage prediction to absorbing at least a year's worth of growth in data that translates into number of instances needed, licensing costs for both production and test environments, using RDS services vs. managing larger EC2 instances, ensuring that database architecture can address scalability, availability and reliability of data. Creating disaster recovery plan, planning minimal migration downtime possible and the list goes on. As part of this migration, we decided to migrate from licenced Oracle to open source MYSQL database running on Netflix managed EC2 instances."

Overall this transformation scope and exercise included:

1. **APIs and Integrations –** The legacy billing systems ran via batch job updates, integrating messaging updates from services such as gift cards, and billing APIs are also fundamental to customer workflows such as signups, cancellations or address changes.
2. **Globalization –** Some of the APIs needed to be multi-region and highly available, so data was split into multiple Cassandra data stores. A data migration tool was written that transformed member billing attributes spread across many tables in oracle into a much smaller Cassandra structure.
3. **ACID –** Payment processing needed ACID transaction, and so was migrated to MySQL. Netflix worked with the AWS team to develop a multi-region, scalable architecture for their MySQL master with DRBD copy and multiple read replicas available in different regions, with toolingn and alerts for MySQL instances to ensure monitoring and recovery as needed.

4. **Data / Code Purging –** To optimize how much data needed migrated, the team conducted a review with business teams to identify what data was still actually live, and from that review purged many unnecessary and obsolete data sets. As part of this housekeeping obsolete code was also identified and removed.

A headline challenge was the real-time aspect, ‘changing the tyre of the moving car’, migrating data to MySQL that is constantly changing. This was achieved through Oracle GoldenGate, which could replicate their tables across heterogeneous databases, along with ongoing incremental changes. It took a heavy testing period of two months to complete the migration via this approach.

Downtime Switchover

Downtime was needed for this scale of data migration, and to mitigate impact for users Netflix employed an approach of ***‘decoupling user facing flows to shield customer experience from downtimes or other migration impacts’***.

All of their tooling was built around ability to migrate a country at a time and funnel traffic as needed. They worked with ecommerce and membership services to change integration in user workflows to an asynchronous model, building retry capabilities to rerun failed processing and repeat as needed.

An absolute requirement was SOX Compliance, and for this Netflix made use of components available in their OSS open source suite.

"Our Cloud deployment tool Spinnaker was enhanced to capture details of deployment and pipe events to Chronos and our Big Data Platform for auditability. We needed to enhance Cassandra client for authentication and auditable actions. We wrote new alerts using Atlas that would help us in monitoring our applications and data in the Cloud."

Building HA, Globally Distributed Cloud Applications with AWS

Netflix provides a detailed, repeatable best practice case study for implementing AWS Cloud services, at an extremely large scale, and so is an ideal baseline candidate for any enterprise organization considering the same types of scale challenges, especially with an emphasis on HA – High Availability.

Two Netflix presentations: Globally Distributed Cloud Applications, and From Clouds to Roots provide a broad and deep review of their overall global architecture approach, in terms of exploiting AWS with the largest and most demanding of of capacity and growth requirements, such as hosting tens of thousands of virtual server instances to operate the Netflix service, auto-scaling by 3k/day.

This goes into a granular level of detail of how they monitor performance, and then additionally in they focus specifically on High Availability Architecture, providing a broad and deep blueprint for this scenario requirements.

Cloud Native Toolchains

Migration is only the first step of the journey. Netflix has achieved their massive global growth because they are innovating continuously, meaning that they must be continually updating and changing their digital business systems.

This introduces the role of Continuous Deployment best practices, and how one of their modules 'Spinnaker' is central to this.

In this blog Global Continuous Delivery With Spinnaker they explain how it addresses this scope of the code development lifecycle, across global teams, and forms the backbone of their DevOps 'toolchain', integrating with other tools such as Git, Nebula, Jenkins and Bakery.

As they describe:

> Spinnaker is an open source multi-cloud Continuous Delivery platform for releasing software changes with high velocity and confidence. Spinnaker is designed with pluggability in mind; the platform aims to make it easy to extend and enhance cloud deployment models.

Their own quoted inspirations include Jez Humble's blog and book on Continuous Delivery, as well as experts such as Martin Fowler and working ideals such as 'Blue Green Deployments'.

Moving from Asgard

Their history leading up to the conception and deployment of Spinnaker is helpful reading too; previously they utilized a tool called 'Asgard', and in Moving from Asgard:, describe the limitations they reached using that type of tool, and how instead they sought a new tool that could achieve:

- *"enable repeatable automated deployments captured as flexible pipelines and configurable pipeline stages*
- *provide a global view across all the environments that an application passes through in its deployment pipeline*
- *offer programmatic configuration and execution via a consistent and reliable API*
- *be easy to configure, maintain, and extend"*

These requirements formed into Spinnaker and the deployment practices they describe, which you can repeat through the Github Download.

Microservices

The third Cloud Native foundation component is a microservices software architecture and again there are a wealth of resources to learn from.

In the past, architects tried to design every aspect of a software. Applications were supposed to work like perfect machines. The microservices approach to software development is more organic. Instead of trying to control every aspect of a complex system, architects try to set up rules to make a functioning organism. DevOps tools play an important role in this ecosystem. These tools help multiple teams work with each other seamlessly. The process results in healthy, flexible and scalable software.

Netflix OSS is a great place for any DevOps team to get an idea of the variety of tools that Netflix uses to run its massive microservice-based applications.

Matias De Santi of Wolox describes how microservices can make use of AWS services like their API Gateway and RisingStack offers this article Deploying Node.js Microservices to AWS using Docker.

- **Build and Delivery Tools:** Netflix has a collection of Gradle plugins called Nebula that helps development teams create repeatable builds. It helps save time during development. The Aminator tool packages AMIs for AWS. Spinnaker is Netflix's continuous delivery platform that makes its complex microservices deployment possible.
- **Common Runtime Services and Libraries:** Eureka is Netflix's service discovery tool. The application Ribbon helps with service communications. Hystrix helps isolate latency and fault tolerance at runtime.
- **Data Persistence:** Netflix's EVCache and Dynomite are important innovations that help microservices use Memcached and Redis at scale.
- **Insight, Reliability, and Performance:** Netflix has developed a lot of tools to collect metrics and automatically address problems. But Chaos Monkey and Simian Army are its most famous reliability tools. These tools help Netflix test instances with random failures. The above discussion touches only a handful of options. The Netflix OSS page has a list of all the available open source tools that can help DevOps practices.

Microservices at Netflix Scale - Lessons Learned

This insightful 2016 talk about Microservices at Netflix was presented by Ruslan Meshenberg who served as the Director of Platform Engineering at Netflix.

Netflix operates a platform approach that other developers utilize and build upon, a common layer that enables developer teams to work autonomously and quickly. The colossal levels of downstream traffic is supported by over 500 microservices, with 100 to 1,000 production changes being deployed daily.

Monolith to Microservices Cloud Migration

From *3:30* Ruslan describes the Netflix journey to the Cloud, with their desire to migrate being the primary motivation for also adopting microservices. It wasn't possible to simply lift and shift their on premise monolith application, instead they 'chiseled' it up into microservice components and deployed these to the Cloud, a process that ultimately took seven years to complete.

At *5:35* he explains the primary catalyst for this move was a major failure of the legacy application, a database corruption that took four days to recover and caused a severe customer impact. This prompted their move to become an entirely Cloud-based company.

Core Principles

From *6:20* Ruslan begins to explain their microservices architecture, defining the core principles for guiding their use.

- **Use (and contribute to) open source where possible –** Only develop new code if you really have to.
- **Services should be stateless –** Don't rely on sticky sessions, except for persistence and caching layer, and prove through chaos testing.
- **Scale out vs scale up –** The key dynamic of the Cloud is the ability to provision new instances in response to escalating demand.
- **Redundancy and isolation for resiliency** – Make more than one of anything, and isolate the 'blast radius' of any one component failure.
- **Automate destructive testing –** Assume failure will occur and be continually testing the system to prove it can withstand those failures.

From 11:00 Ruslan moves on to exploring how they apply these principles in action, such as their use of Chaos Monkey for automating failure testing, and in particular their implementation of the Cassandra database, and how best to optimize large scale and multi region data consistency.

At *15:30* he addresses the billing component, the last and most intense of their microservice migrations, as they are naturally critically important and are also exposed to strict compliance requirements, necessitating logging and auditing.

Microservices – Benefits

From *16:30* Ruslan moves on to defining the benefits of a microservices approach.

These benefits are best articulated through relating them to the organizations priorities; in the case of Netflix they ranked 1) velocity of innovation, 2) reliability and 3) efficiency, as their core goals.

What is most notable about this paradigm isn't the technology but organizational models. Ruslan highlights that the core dynamic of monolithic enterprise systems that inhibits innovation is the single software deployment life-cycle. Yes there are multiple teams working on various innovations, but each must be processed into and through this life-cycle, a "tight coupling" between the teams.

"Loosely coupled" teams work independently of each other and operate an end-to-end ownership of their own services, there is no central stage gate, they each manage the releases of their own code.

Furthermore their platform approach provides each team the building blocks, addressing core needs like security, for them to build upon, a 'separation of concerns' approach.

Microservices – Costs

From *19:40* Ruslan articulates the costs of the microservices approach. Again this comes back not to technology but organization – The new autonomous teams approach requires a culture and structure change, for example there is no longer a single QA team, and adapting to these changes presents human challenges.

The main cost factor they experienced was the dual operation of both on premise and Cloud Native platforms while they migrated. Understandably this presented a huge workload and double complexity expense in all areas such as bugs and maintenance.

However having made that investment they have empowered themselves to become the global digital behemoth success story we know today.

Microservices – Lessons Learned

So what lessons did Netflix learn from this journey? From *23:05* Ruslan explains that primarily these were:

- **IPC is crucial for loose coupling –** A common language for communications and contracting between microservices.
- **Common deployment methods –** A homogeneity of how applications are deployed. Netflix initially achieved this through Asgard, then moved to Spinnaker. This provides a single source of truth for managing roll outs.
- **Database caching –** Hundreds of microservices interacting with the databases presents a significant challenge, one that Netflix protected against through implementing a caching layer, using EVCache.
- **Automated telemetry –** Netflix generates over 20 million metrics per second, adding up to 1.7 trillion a day. Thus human processing of this vastness of reporting is impossible, and so these are piped into automated error detection and remediation algorithms.

At *28:20* Ruslan highlights the core challenge this presents for traditional approaches to enterprise IT architecture, notably the pointlessness of maintaining enterprise architecture diagrams, as these are rendered obsolete almost instantly. Instead the business needs a real-time view, achieved through this automated telemetry.

Reliability

At *29:30* he moves on to the critical importance of reliability and how this is achieved.

The key to this is primarily the assumption that failures will occur and therefore the system is engineered with a capacity to rapidly adapt to them. Netflix pioneered and operates an approach of 'circuit breakers', implemented through their Hystrix component.

Again automation of these adaptations is critical as is the ongoing destructive testing of the environment to ensure the capability. They implement these tests at a very large scale, simulating the failure of entire AWS regions, demonstrated in visual form at *35:35*.

Containers and Conclusion

From *36:35* Ruslan focuses on the role of containers, highlighting they provide a valuable but not silver bullet tool for implementing microservices.

As a very early Kubernetes user they had to develop many of the additional components they needed, and wrapping up he summarizes these and other tools that are available from Netflix.github.com.

Mastering Chaos - A Netflix Guide to Microservices

A complement to the Microservices at Netflix Scale talk is this presentation from Josh Evans: *Mastering Chaos – A Netflix Guide to Microservices*.

Josh provides a deeper dive into the detail of the architecture components. Download the slides & audio at InfoQ.

Introduction and Overview

Beginning at *5:30* Josh provides an introduction to what microservices are and aren't.

He starts with the basics- the anatomy of a microservice, the challenges around distributed systems, and the benefits. Then he builds on that foundation exploring the cultural, architectural, and operational methods that lead to microservice mastery.

At *6:02* he states that Netflix initially had an infrastructure that was hardware oriented and very expensive. He also added that so many changes were happening on a regular basis to the base architecture.

At *8:22* Josh defines microservices to be an architectural approach for developing the single application as a suite of the small services ensuring that each run its own process and communicates via lightweight mechanisms. At *9:38* he emphasizes that separation of concerns that include modularity, encapsulation and the biggest benefits of Netflix microservices that includes scalability and virtualization.

Edge Service

From *10:00* Josh walks through the overall architecture of the Netflix platform.

He starts with the Edge Service, made up of the ELB and Zuul for dynamic routing, the NCCP (Netflix Content Control Plane) and the API gateway that is core to their modern architecture, calling out to all the other services to fulfill customer requests.

At 10:40 he moves on to the middle tier and platform, an environment made up of many components such as an A/B testing and subscriber service, a recommendations system and platform services such as microservices routing, dynamic configuration, crypto operations and persistence layers.

Microservices architecture and practices

From *11:40* he explores the core principles and challenges of microservices, highlighting the complexity of their inter-operation with client libraries and caches, and explores mitigating these challenges through four key factors of Dependency, Scale, Variance and Change.

At *12:55* Josh explains the way microservices achieve abstraction, the way the EVCache Client sends the request which is then processed in the backend database with the help of the server.

At *15:10* he explains an interesting scenario of Cascading Failure wherein one can find one service failing with the improper defenses against the failing service, it cascades which would, in turn can then demolish the entire deployed system.

Josh elaborates on the role of Hystrix-based ‘circuit breakers’ in preventing this from occurring, at 16:40 he describing the practice of FIT – Failure Injection Testing that this enables. As the name suggests this provides a method for testing various microservice scenarios within a live context.

Critical Microservices

A challenge that this presents is how to manage the scope of what is tested, given the exponential scale of permutations that can arise from so many microservice interactions.

Central to achieving this is that Netflix defined ‘Critical Microservices’ – The minimum level of service a customer would want in the event of failures, ie. a basic browse and watch capability. Customers would be accepting of the loss of some of the value add services like personalization as long as they can achieve at least this.

From this they created ‘FIT recipes’, templates that blacklisted all the other non-critical services. This enabled them to test the ongoing availability of critical services in the event of the loss of the other functions.

Return of the Monolith

From *19:15* Josh explains the heated debate Netflix had about whether to adopt an approach of building client libraries or not.

They perceived considerable benefit to them, through the availability of common logic and access patterns for calling services as a much more simplified approach and thus decided upon them.

The interesting challenge that arose is that in essence this began rebuilding a monolith application, a new kind where the API gateway is running a lot of in-process code, giving rise to problems like heap consumption, logical defects and transitive dependencies that pull in conflicting libraries.

The conclusion to this debate has been a goal of the balance between the ultra simple, bare bones REST only model and achieving the most simplified client libraries possible.

Eventual Consistency

Beginning at 22:00 Josh explores their approach to persistence, with the 'CAP Theorem' guiding their thoughts in this area, wherein one will have to choose between consistency and reliability, in scenarios where one particular database in an availability zone may not be available while the others are.

Netflix opted for an approach of 'Eventual Consistency', the choice where you write to the others and catch up later with the full replication between them to achieve data consistency, a feature that Cassandra works very well for.

Scale

At *25:00* Josh moves on to a major section discussing scaling.

This is broken down into different sections addressing stateless and stateful service scenarios. He first explains that through the use of auto-scaling dealing with the scaling requirements of stateless services is almost a no-brainer. Responding to failures and traffic spikes is easily accomplished through auto-provisioning new AWS resource.

Stateful services, those that use databases and/or store significant levels of their own data, are a different ball game all together, presenting a much more significant challenge for handling their scaling requirements.

Technology central to this scenario is the sharded use of EVCache. Not only does this write data to multiple nodes but across multiple availability zones too. Reading from them happens locally but again can the application can read from across multiple zones if needed.

At *31:40* he moves to the current trending topic of hybrid microservices wherein excessive load scenarios can be managed well with the hybrid architectural design. He adds on that workload partitioning and the request-level caching technique are employed for hybrid cases.

Variance

At *33:35* Josh explores the challenges of variance within an IT architecture, a challenge that grows in scale as it in increases, primarily through operational drift and the introduction of new languages and containers.

Operational drift is the inevitable aging of different factors of maintaining a complex system that happens unconsciously, best addressed through continuous learning and automation. For example quickly learning from incidents as they happen and automating the remedying best practices into the infrastructure, so that “knowledge becomes code”.

In contrast introducing new languages and technologies like containers are conscious decisions to add new complexity to the environment. His operation team standardized a ‘paved road’ of the best technologies for Netflix that baked in their pre-defined best practices, based around Java and EC2, but as the business evolved developers began adding new components such as Python, Ruby, Node-JS and Docker.

This repeated the challenge of growing a new monolith, with the API service becoming overloaded with code in such a way as to cause various failure scenarios; they addressed this through sculpting out Node-JS components to run as small apps in Docker containers.

At *40:15* Josh summarizes the cost of these variances, notably:

- **Productivity tooling –** Managing these new technologies required new tools.
- **Insight and triage capabilities –** A keynote example is new tools are needed to reveal insights about performance factors.
- **Base image fragmentation –** A simple base AMI became more fragmented and specialized.
- **Node management –** The challenge of node management was so significant they found there was no available off the shelf technology for it, so they built Titus for this.
- **Library / platform duplication –** Providing the same core platform functions to these new technologies required duplicating it to the new tools, such as rewriting some of them in Node JS.
- **Learning curve / production expertise –** Inevitably introducing new technologies presents new challenges that must be overcome and learned from.

Ultimately this resulted in them operating multiple ‘paved roads’, a variance which they sought to minimize and manage through constraining centralized support, educating teams as to the costs of their decisions and where possible to seek reusable solutions.

Change velocity

From 43:30 Josh begins to wrap up by examining the headline theme – How do you achieve software innovation velocity with confidence? How can you be continually introducing new change into a system with minimal breakages?

Fundamentally Netflix achieved this through their use of Spinnaker. This integrated the best practices they had learned into their deployment life-cycles, automatically applying capabilities such as canary analysis and staged deployments.

Closing out he goes back to 2009 to describe the evolution of the Netflix tech organization, with a view to explaining how departmental structures and dynamics can also play a major force in shaping the design of systems and be a factor in enabling or inhibiting change.