Tech Mahindra

Cloud Native Applications and the new paradigm of Quality Engineering

Testing for Recovery

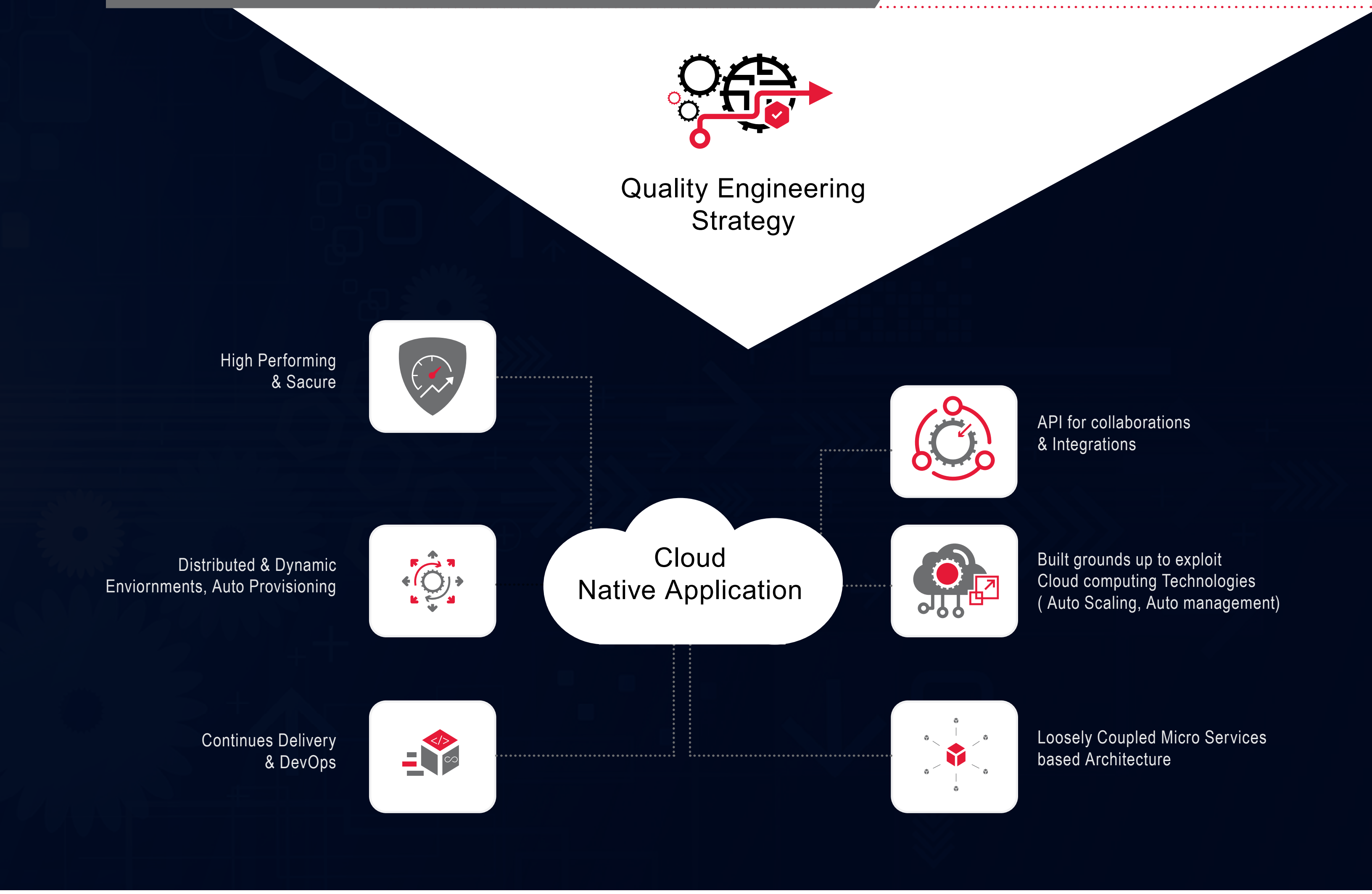
Authored by
Anjali Chhabra Nandwani

Connected World.
Connected Experiences.

Executive Summary

Technology leaders across industries are setting up cloud native application development teams focusing on delivering cloud native offerings to meet critical business requirements stemming from Digital Transformation initiatives. These teams need to strategize on the crucial aspects of continuous delivery and quality engineering to ensure high quality deliverables at high velocity. This thought paper presents the point of view on Quality Engineering Strategy aligned with Continuous Delivery for Cloud Native Application Development

Cloud Native Applications and the new paradigm of Quality Engineering i.e Testing for Recovery


Quality Engineering Strategy
High Performing & Sacure
Distributed & Dynamic Enviornments, Auto Provisioning
Continues Delivery & DevOps
Cloud Native Application
API for collaborations & Integrations
Built grounds up to exploit Cloud computing Technologies (Auto Scaling, Auto management)
Loosely Coupled Micro Services based Architecture

To start with let's level set that why this article on Quality Engineering for Cloud Native applications

What does Gartner analysts say about Cloud-Native -

"Technology leaders must deliver cloud-native offerings now to capture business opportunities and avoid irrelevancy"

Craig Lowery
Gartner Principal Analyst

As per a latest publication on technology predictions by 2022,

50-60% of traditional workloads and applications being moved to cloud will be re-written using cloud native architecture guidelines.

A question raised frequently by Customers and Peers in the Industry

'What is different when it comes to defining quality engineering strategy for a cloud native development? Or is there really any difference?'

I always try and share my perspective, as to what we, in Digital Assurance Services team at Tech Mahindra, believe is Unique and how we are testing to enable continuous delivery for cloud native applications. This paper will present my observations learnt from many customer engagements and interactions with industry teams in various forums.

Now, to understand a bit more on Testing for Cloud Native Applications, it is important to first understand what exactly a Cloud Native app is **Native Cloud applications are the applications that are designed and built grounds up to exploit the advantages of cloud computation technologies;** where IaaS (Infra as a Service) capabilities of providers like Azure, AWS, Google cloud can be utilized to provide auto scaling, auto management for applications, leading to optimized resource consumption and reduced operational costs. These applications are l**oosely coupled, built mostly using micro services and twelve factor principles and that run in containerized fashion on dynamic environments** with a combination of on premise environments, private, public & hybrid cloud platforms. Another important aspect is that, mostly (almost always) these applications **use Continuous Delivery model,** backed up with Agile and DevOps principles. Also, some more keywords and phrases that can be used to define these apps are **Scalable, Maintainable, Resilient, High Performing, Secure and Easy to change for high speed delivery of new features.**

In a nutshell, if I have to create my own definition, then **"Native Cloud Applications are designed and built grounds up to exploit cloud computing technologies to deliver high performing, scalable, secure and resilient applications; capable of adapting to distributed and hybrid environments, to meet 'velocity with high quality' demands of business users in a very demanding Digital Economy."**

Now, for a much more formal definition, I would suggest to refer the one from the "Cloud Native Computing Foundation (CNCF)". I am not copying it here but there is one from CNCF available.

Also, **one small but important point is that migrating monolithic on premise applications to IaaS cloud does not make them cloud native, neither does adding APIs to monolithic applications make it cloud native; monoliths are to be refactored, re-architected, and functionally decomposed (so almost re-developed) to be service oriented, distributed and scalable on demand, to be called cloud native.**

CHARACTERISTICS OF NATIVE CLOUD APPLICATION

Why it is important to understand the characteristics of native cloud applications, is because some of these characteristics eventually drive those finer details and some differences in the Quality Engineering principles and test strategy for these applications.

Let's start reviewing the characteristics of native cloud applications, one by one, and see what QE teams need to do to meet the expectations. Also, I will limit this article more to the strategy aspects without getting into tools/technologies for implementing those strategies.

Designed and built grounds up - backed up with Agile and DevOps Principles

This is an important characteristic that drives the process for quality engineering, to a great extent. One common issue from QE perspective (irrespective of who is responsible for QE – Full Stack Teams, SDETs, Functional QE group etc.) is the right quality engineering thought process in the initial phases. However, this characteristic of Native Cloud Applications provide the right opportunity to think through all aspects of quality, by embedding the right story level acceptance criteria in 'Definition of Ready (DoR)' and Sprint level acceptance criteria in 'Definition of Done (DoD)'.

Also considering that it is grounds up development, the teams generally focus on starting with delivering a 'Minimum Viable Product (MVP)' that gives opportunity for teams to pick up features that can create the first MVP delivery and plan for lifecycle quality engineering principles including the functional and non-functional aspects.

While it is comparatively easy to create functional acceptance criteria, one key attribute for Native Cloud Applications is always the Non-Functional acceptance criteria, which is not easy to capture.

Considering distributed and dynamic environments, there can be multiple different scenarios of Failures due to Non-Functional aspects, and as it is considered nearly impossible to document each and every such scenario, from service failures to network issues, latency, data packet loss and so on, QE teams are supposed to "Test for Recovery", how recoverable the application (along with the data and state of application) is, in case of an unknown and undefined failure, when it comes to Non-Functional scenarios –. We will touch upon this more while covering the Resilience characteristic, as this is an important QE principle to define the Non-Functional acceptance criteria and our "Test for Recovery" strategy.

Just to summarize this point though – it is important to have enough details in DoR and DoD for acceptance criteria on both functional and non-functional aspects and focus on delivering MVP sooner than later, with the capability to build on MVP with more and more features.

One more aspect around Continuous Delivery and CI/CD pipelines – while this can be another detailed subject in itself; however to ensure robust Cloud Native app delivery with high-quality, CI/CD pipeline / platform is to be treated as any other software and has to be maintained with the same level of discipline and same rigor of quality engineering principles are to be applied here.

Another critical aspect to consider is the **Delivery Model shift** for Cloud Native developments; while it does not contribute directly to the technical aspects of Cloud Native applications it still impacts the Quality Engineering strategy.

So **the shift from traditional ticket driven to more API and event driven delivery model along with more decentralization and self-service driven delivery models need to be considered while defining the QE strategy.**

Loosely Coupled Applications and Microservices Based Architecture

Micro services based architecture helps break monoliths and increases agility of development. Ideally, the goal of Microservices architecture is to ensure that each service is just one feature and thus, Unit Testing becomes an important aspect in this case. Again, though the ideal world is one feature-one service, the reality is not always that, so a set of features can still become a micro service and in that case, component testing needs to be in the strategy.

Contract Testing strategy

Contract Tests are an important part of Micro services testing strategy which ensure that services honor their API contracts and these contracts do not break with changes/ enhancements. In complex scenarios, most of the times, one user request generally traverses through a set of micro services to respond back to the user, that means, integration testing of service interfaces/req-res becomes a critical aspect of the testing strategy.

Also, **in real time scenarios, having an ability to trace the user request to the service navigation then back to the user with the response, becomes critical for a quick resolution of issues. There are multiple open-source tools that provide specific ability to break a single request into service-to-service tracing type of view for better "Observability".**

Observability is one other key aspect of Native Cloud Applications and should be considered by the Quality Strategist of a Native Cloud Application as an area to focus on while testing the app to find out: how observable the app is, how searchable and meaningful the logs are and, can all events be traced properly if the system is event driven.

Also, many modern cloud native architectures are adopting Service Mesh for microservices runtime infrastructure for ease of service discovery, observability, security and load balancing capabilities. So, the service mesh adds to another consideration while defining the quality engineering strategy.

Distributed and Dynamic Environments

This is a complex subject in itself; however, **from a QA strategy perspective it is important to understand certain aspects of the application - is it deployed across cloud providers or a hybrid environment? How a business critical feature needs to be recovered, if a part of distributed environment fails?**

For e.g. if app server and databases for the Software in question are across two providers, what happens if the database provider faces an outage? Will the app server be automatically routed to some cached data? What will be the impact on user experience if that happens? This is just one scenario; however, QA strategists, along with other relevant stakeholders, need to think of many such scenarios and device test strategy and recovery strategy for such scenarios.

High Performing and Scalable

Scalability has always been a consideration of a good system architect. It is one of the critical characteristics, and not just a requirement of Native Cloud Applications from QA strategy perspective; so validation of scalability is important.

How do services scale?
Let's understand that Scalability of a Micro Services based application is a bit different as compared to traditional monolithic architectures.

If a monolith app running on a single server has scalability challenges–it is easy to add more resources for the application to resolve the problem. However, if an application with micro services architecture in a distributed environment runs into scalability issues, it is not as simple to allocate more resources.

The challenge is to identify the aspects of resource allocation– where to allocate more resources? And for which component? That is why, testing upfront for performance and scaling and benchmarking performance of all services and components is a very important aspect so that if something hits production, we know (through monitoring) by comparing benchmarks, which component needs upscaling. Most of the times, required components are set-up for auto scaling.

Also, while testing for performance, i.e., for Load, Stress and Soak, each area is be considered and tested for. And obviously, recovery mechanisms need to be defined and implemented upfront.

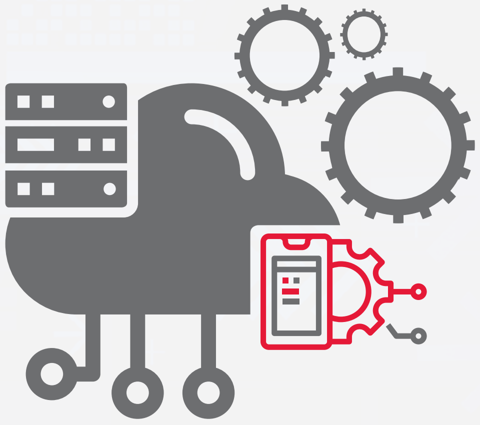
I cannot emphasize enough on the importance of Unit and Component level load and stress tests for micro services which have to be a part of DoD.

Resilient

Again, the idea of cloud native development, is also to consider a design and development strategy that decouples them from servers and operating system dependencies, **so by the design, cloud native apps are a collection of light weight and containerized services.** For integrations and collaboration, cloud native architectures depend on APIs, and also the expectation from these apps, is to align with the elastic infrastructure, that can be scaled up or down dynamically, based on the load situation.

So, the point here is that, by design itself, cloud native applications are meant to be highly resilient, however, all these aspects need to be covered as part of the test strategy, for ensuring that the design principles are successfully implemented.

Chaos Engineering is one of the strategies that is to be closely looked at and incorporated for ensuring the resiliency aspects, for example, bringing down one containerized micro service randomly and analyzing the impact on overall user experience and then subsequent recovery.



Secure

A recent Gartner publication claims that by 2022, API abuses will be the most frequent attack, resulting in data breaches.

Hence security testing strategy, in my view, is one of most important consideration for the cloud native application testing strategist.

Security Testing is a topic that can have a detailed article on its own, however, in this section, let us focus on what is the core difference between security testing strategies for a regular application versus native cloud applications. By design, the native cloud applications are service based architectures and also the idea is to have micro services that can be consumed using APIs, while each micro service will have at least one API. It is not necessary that it has only one API to consume the service; however, there can be different approaches by which the consuming applications can consume a micro service.

The point is that for a traditional monolithic application, it is easy to determine and authenticate its consumer, but that is not the case for native cloud applications, so the traditional security strategy that works well for monolithic applications, like securing the end points as well as agent based security techniques, potential vulnerability alerts etc. are to be relooked at.

The end-to-end security monitoring concept gets highly diluted as we move towards micro services, as the environments can be dynamic and alert volumes can quickly grow to become unmanageable.

As the delivery of cloud native applications needs to be continuous, light weight and automated along with CI/CD pipeline.

Also, it has to incorporate some principles of chaos based testing to manage distributed and dynamic environment aspects.

Quality engineering for intrusion detection systems itself is a key aspect. Traditional detection of security threats can be a very manually exhaustive process, including analysis of huge volume of security alerts, leading to huge efforts and delays in response to a real threat. Continuous Delivery needs faster detection and response.

Quality engineering for intrusion detection systems itself is a key aspect. Traditional detection of security threats can be a very manually exhaustive process, including analysis of huge volume of security alerts, leading to huge efforts and delays in response to a real threat. Continuous Delivery needs faster detection and response. Also, a poorly configured detection rule can create false alerts, leading to confusion and more effort with low results. For cloud native app economy, the detection has to be modern and engineered with automated classification of alerts and auto response mechanisms. **Distributed environment means the detection workflows may cut across containers, VMs, hybrid cloud environments, so it is important to test even the detection workflows for reliability and quality. Automated detection categorization and review/response also means that a lot of action can happen without human reviews, so the quality engineering aspect for the detection system itself, is to be reviewed and enhanced.**

How challenging it can be, for an end user, if it is forced to change the password at every second login, due to a poorly tested detection workflow that is invoking a (false) threat rule, which in turn is routing the program to change password routine, which is created to overcome the threat!

So, with the very interesting characteristic of security, I think I will close this article. There is a lot that we can discuss on test strategy for Cloud Native Applications but I have tried to cover all the key aspects that we, as Tech Mahindra's Digital Assurance practice, cover.

The interesting aspect is when it comes to implementation of these strategies; we have our IP LitmusT (AI powered intelligent automation platform) that has accelerators to cover all of these aspects, from implementation and execution perspective. Do reach out to learn more on how LitmusT can accelerate your journey to rapid Cloud Native Application development.

Anjali is a technology leader with 20 years of testing & QA leadership experience in Information Technology Services industry, with specialization in optimizing Software Development Lifecycle (SDLC) and Quality Assurance Services.
She has worked with large enterprise customers ranging from Banking and Financial Services, Telecommunication, Technology to various verticals. She has proven track record on DevQAOps Transformation initiatives, Consulting for DevOps and Agile Transformation initiatives, Automated Delivery Pipeline (CI/CD/CT) framework design and set-up for large enterprises, QA Community of Practice (COP) set-up, DevQAOps Program Management and Governance. In her role, she has worked with diverse IT teams globally to optimize SDLC with usage of Predictive Analytics, Machine Learning and RPA. She comes with in-depth understanding of Integrated SDLC Automation experience along with framework design / tool selection and ROI analysis.

Anjali Chhabra Nandwani
VP & Head Digital Assurance Services
Tech Mahindra Americas
LinkedIn

Tech Mahindra

www.techmahindra.com
connect@techmahindra.com
www.youtube.com/user/techmahindra09
www.facebook.com/TechMahindra
www.twitter.com/Tech_Mahindra
www.linkedin.com/company/tech-mahindra