Container Solutions

CLOUD NATIVE PATTERN LANGUAGE

Pini Reznik

ABOUT THE AUTHOR

Pini Reznik is a long time developer and software architect. His strong interest in distributed systems and development methodologies is coupled with a keen willingness to surf along with the fast-evolving technologies of cloud computing.

As co-founder and CTO of Container Solutions, Pini oversees the architecture and implementation of Cloud Native migrations for organisations from every sector. His work spans two decades in the configuration management field, with current emphasis on DevOps, automation and Cloud Native CI/CD. He enjoys solving the challenges presented by emerging technology and, working with partners such as Cisco, Google and others. Pini recently became fascinated with the pattern languages and evolutionary design that help to collect and share knowledge in order to build complex systems in a series of small incremental steps.

Container Solutions

As experts in Cloud Native strategy and technology, Container Solutions guide their clients through migrations to the cloud. Their approach is founded upon careful analysis of customer needs. Then, together with your team, they design and implement custom solutions that will last. In-depth education and training are core components of a CS-led transformation, ensuring that your team operates with confidence in the cloud.

Container Solutions' diverse team is equipped with a broad range of Cloud Native skills, with a focus on distributed system development. Container Solutions have global perspective, with offices throughout Europe, in the UK, and Canada.

CLOUD NATIVE TRANSFORMATION PATTERN LANGUAGE

Pini Reznik

September 15, 2018

Abstract

Cloud computing offers tremendous opportunity to develop, deploy and update software faster than ever before. But if companies with older, pre-cloud systems simply shift operation to the cloud they obtain only minimal benefit. Maximizing the advantages of cloud infrastructure requires significant redesign of both organisational systems and culture. Cloud Native architecture emerged to support this transformation. Cloud Native itself is, however, very complex and people find it difficult to understand and use. A Cloud Native Pattern Language will create a set of patterns behind Cloud Native architecture and form a clear way to describe the system. This will allow engineers, developers and executives alike to discuss, disseminate and apply best practises in Cloud Native. In this document we will examine some of the Cloud Native design patterns that we've learned in the course of three years of guiding enterprises onto the Cloud as well as the contexts where they perform best.

INTRODUCTION

Cloud Native is a methodology concerned with architecture, design, process, infrastructure and organisational culture to help enterprises achieve optimal performance in the cloud. Though a still-emerging and rapidly evolving design philosophy, we now have enough examples of good design to begin defining what Cloud Native approaches work best, and in which circumstances. These examples can be used as the basis for creating a set of context-specific Cloud Native design patterns that will form a clear way to describe the system.

THE CLOUD COMPUTING PROBLEM

The problem with cloud computing is that companies cannot simply 'lift and shift' their legacy operations onto the cloud. Migrating without altering the existing organizational structure and development/delivery processes to suit this new environment is an ineffective strategy. The result of doing so is, essentially, to create an expensive new data center on the cloud while failing to access many of the promised benefits of being there in the first place.

Cloud Native exists to help companies take advantage of everything the cloud has to offer. It does this by acting as a model, guiding crucial decisions about technology and culture to best harness cloud benefits. Even as Cloud Native helps solve the cloud computing problem, though, it introduces a new one: Cloud Native itself is very complex, due to the inherent complexity of distributed systems. Patterns addressing this, however, help solve the problem of complexity.

WHAT IS CLOUD NATIVE?

Cloud Native is the name of a particular approach to designing, building and running computer applications. The objective is usually to increase the speed of application development and delivery, that is, getting a new idea into production within days or even hours, instead of months.
Most enterprises migrating to Cloud Native cite velocity as their primary motive.

Ten years ago, the Financial Times of London faced a dilemma. The steep decline of the physical newspaper business model plus the burden of legacy infrastructure were detrimental to the company's survival. However, no one knew exactly what the future of print publication would look like. The FT's board recognized that, to not merely survive but truly prosper in the digital age, the company needed to embrace cloud technology and that this would require a complete restructuring.

The FT solved their dilemma by being culturally flexible and open to change in order to adapt to an increasingly online world. The company worked to simultaneously migrate their business operations to the cloud and to create a system supporting the rapid delivery of innovative digital publishing products. They succeeded by embracing Cloud Native design principles: a Microservices-oriented architecture delivering containerised applications via virtualized (cloud) infrastructure. As a result the FT are able to rapidly and continually develop new features and bring them quickly to market, and are now recognized as a pioneer in digital publishing.

The FT's transformation strategy embraced the three foundations of Cloud Native technology: microservices, containerisation, and cloud-based infrastructure.

- **Microservices** are used to build a whole application from a collection of smaller services, each handling a different function or utility and then harnessed together. This modularity makes the application faster and easier to develop, test and release. "Decomposing" an application into a modular set of services also makes it simpler to understand.
- **Containerisation** encapsulates an entire application into a single package, including its operating system and all dependencies (like the different libraries and configuration files needed to run it). A containerised application is entirely self-contained, secure, and transportable, moving easily from developer's desktop to test environment and on into production.

- **Cloud services**, or Infrastructure-as-a-Service, take the components traditionally present in on-premises data centers, such as servers, data storage and networking hardware, and instead provide them via the internet.

So how do you know Cloud Native when you see it? The core of Cloud Native is how we create and deliver software, not where. So when you see an application built and deployed in small, rapid iterations by a squad of independent, compact feature development teams...And those teams are collaborating via an integrated platform that decouples infrastructure while providing automated monitoring and testing...That is when you know you are looking at the Cloud Native approach in action.

DECOMPOSITION

For a long time, software systems were monoliths. A monolithic application is built as a standalone unit, a single large codebase where everything is tightly coupled and mutually dependent. This means any update or change affects the entire system. One small modification on one small part of the application can require building and deploying an entirely new version. (In the same way, scaling one specific function of a monolithic application also means you have to scale the whole thing). The result is a lengthy wait for developers to see the impact of even a single tiny change. Monolithic architecture limits developer agility and impedes the frequency of new deliveries: new releases typically happen annually, after months of preparation and testing.

Microservices solve these challenges by being as modular as possible. In the simplest form, Microservices architecture decomposes an application into a suite of small modular services, each fully deployable on its own and independent of other functions within the application. These decoupled units each have a specific task, for example payment processing or login services, which can be reconfigured or even entirely rebuilt without affecting the rest of the structure. Teams are able to work in parallel, which speeds development. Scalable, testable software can be delivered weekly, even daily, rather than yearly. Enterprises gain the ability to move from idea to actual product in front of customers in the shortest amount of time.

THE DIFFICULTY OF DISTRIBUTED SYSTEMS

The heart of Cloud Native architecture is redistributing the monolith into Microservices. The benefits, however, come with a cost: complexity. Dividing infrastructure into modular, related services makes intuitive sense. But this also means managing many moving parts, including monitoring, storage, how different components are behaving together; defining communications, networking security… the complexity becomes almost exponential as the process moves forward.

Developing a Cloud Native patterns language addresses the complexity inherent to distributed systems, and makes it easier for developers to discuss, learn and apply the best practises for handling it.

PATTERNS IN CONTEXT

You might now expect the assertion that Cloud Native systems are intrinsically “right,” thanks to the many benefits of the architecture. The truth is, Cloud Native isn’t an architectural silver bullet. There is no one Cloud Native design that will work well in every circumstance, and so design patterns must be context-specific. A design that ignores context will almost certainly be a painful one to deliver, and difficult to live with.

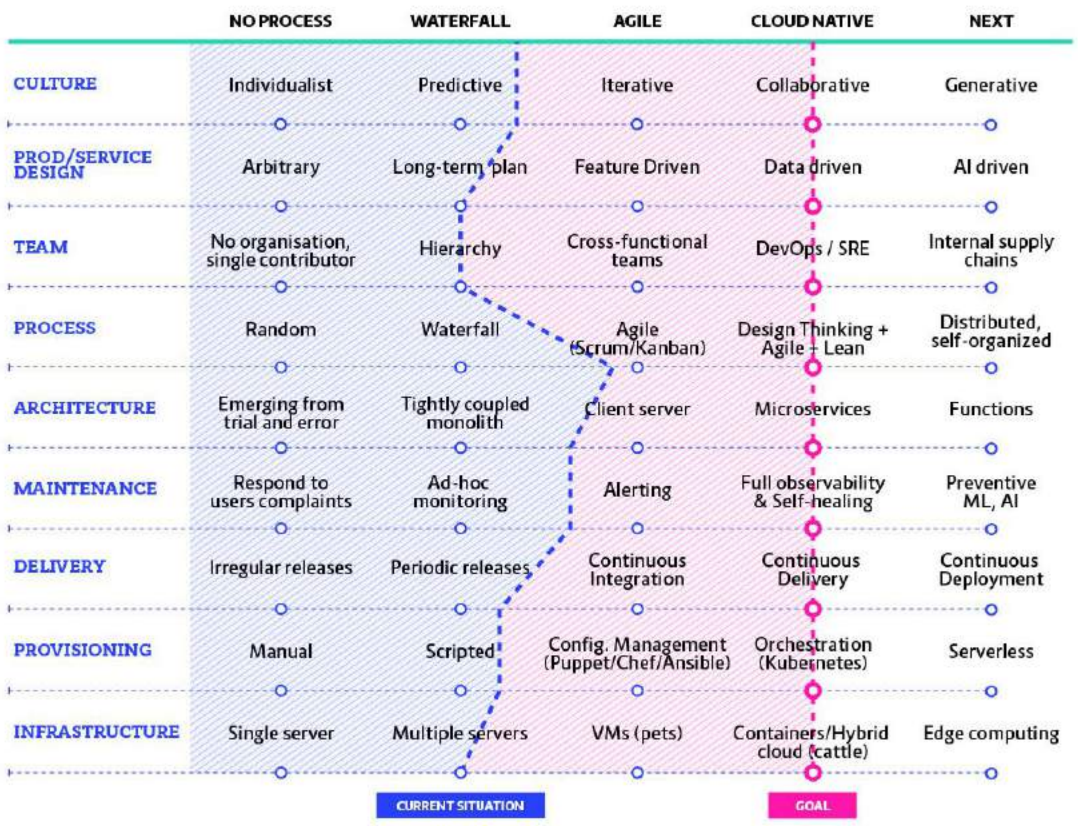
Among the contexts we should consider when making Cloud Native design choices:

- The existing skills of your teams.
- The timescale and goals of your project.
- The internal political situation (how much buy-in is there to a project).
- Budgets.
- Legacy products and tools.
- Existing infrastructure.
- Emotional or commercial tie-in to vendors or products.
- Ongoing maintenance preferences.

Appropriate pattern choices almost always depend on the context where an organisation is at the start of a Cloud Native Migration, as well as its ultimate goals. But how to assess these contexts?

THE MATURITY MATRIX: Context in the real world

Over the past three years, using lessons learned in guiding companies into the cloud, we have developed the Container Solutions Cloud Native Maturity Matrix as an assessment tool. We use it to define, analyse and describe organisational context, both desired and goal, and constantly reassess as the migration progresses. This data allows us to make patterns choices and monitor progress.



	NO PROCESS	WATERFALL	AGILE	CLOUD NATIVE	NEXT
CULTURE	Individualist	Predictive	Iterative	Collaborative	Generative
PROD/SERVICE DESIGN	Arbitrary	Long-term plan	Feature Driven	Data driven	AI driven
TEAM	No organisation, single contributor	Hierarchy	Cross-functional teams	DevOps / SRE	Internal supply chains
PROCESS	Random	Waterfall	Agile (Scrum/Kanban)	Design Thinking + Agile + Lean	Distributed, self-organized
ARCHITECTURE	Emerging from trial and error	Tightly coupled monolith	Client server	Microservices	Functions
MAINTENANCE	Respond to users complaints	Ad-hoc monitoring	Alerting	Full observability & Self-healing	Preventive ML, AI
DELIVERY	Irregular releases	Periodic releases	Continuous Integration	Continuous Delivery	Continuous Deployment
PROVISIONING	Manual	Scripted	Config. Management (Puppet/Chef/Ansible)	Orchestration (Kubernetes)	Serverless
INFRASTRUCTURE	Single server	Multiple servers	VMs (pets)	Containers/Hybrid cloud (cattle)	Edge computing

The Container Solutions Cloud Native Maturity Matrix

It is important to note Cloud Native contexts are not only concerned with technology and software, but also psychological and social aspects. An organisation’s management Process, Team structure, and internal Culture all constituent axes on the

Maturity Matrix are human-centered contexts that hold equal importance to tech-centered ones like Infrastructure and Maintenance/Automation.

Container Solutions have performed a series of case studies on a variety of enterprises like the FT who have built successful Cloud Native systems. From these case studies we pulled real-world examples demonstrating how “the right pattern” can only be the right pattern *in the right context*.

For example Starling Bank, a mobile-only challenger bank founded in 2014. As a startup, Starling had the luxury of being born Cloud Native, using containerised Microservices architecture delivering core processes in the cloud from the company’s very inception.

Starling are both an example of the need to consider context when making architectural choices, and that not every enterprise needs to make identical pattern choices to succeed in the cloud. Conway’s Law states that systems architecture tends to resemble the organisation’s architecture. When it comes to Microservices, many companies follow an architectural approach of assigning responsibility for specific microservices to designated teams, in the same manner that organisational duties are dispersed by department. Starling, however have chosen instead to assign by function, such that every service can be developed on by multiple teams. This pattern choice fits the organisation’s relatively small size and culture of innovation, which allows Starling to reconfigure very quickly and responsively. (In fact, Starling typically re-deploy their entire process multiple times each day). Larger enterprises, however, often benefit from smaller microservices and a more Conway-like model. Starling’s context awareness led to the optimal choice for their specific circumstances, though it was not the most usual pattern applied.

COMMON CLOUD NATIVE CONTEXTS

People don't all apply the same pattern -- they apply the pattern that is appropriate to where they are and where they want to be. Companies coming to Cloud Native from more traditional architectures must assess their initial context and identify their desired outcome. In a Cloud Native patterns language, an enterprise's leaders need to be able to identify their organisation in a specific situation in order to apply the correct patterns for that context. With context identified, patterns can show the forces at work define the problem, and give a solution.

That said, we have observed that many companies looking to commence a migration to Cloud Native share a consistent and typical setup that falls under the Waterfall category on the Maturity Matrix.

Often, they have:

- Traditional Waterfall process with deliveries every few months
- Monolithic applications
- Pre Cloud Native languages (typically Java/C#, but go as old as Cobol)
- Strong, inflexible management hierarchy.
- Little or no automation of infrastructure and development processes

In Waterfall organisations, a complete shift in context is necessary for successful Cloud Native transformation. Monoliths must be broken up into microservices, and automated deployment, testing and maintenance must be put into place. **Most importantly, the organisation must shift its internal culture and hierarchical mindset to become flexible, responsive and above all experimental.** A Waterfall enterprise needs to transform from a culture of stating the top-down "right" answer, to an open approach of exploring and testing many possible answers.

Another common context we find is an organisation that has progressed to an agile approach. As described by the Agile category on the Maturity Matrix, this organisation will have:

- Cross functional teams
- Scrum process
- Microservices architecture (few monoliths in this column)
- some Continuous Integration
- some automation of infrastructure

In Agile organisations, the most natural way forward in a CN migration is to make the transformation tasks part of the teams' technical backlog. However, this approach typically leads to poor results due to the fact that most common implementation of Scrum processes are very much focused on speed of feature delivery. The Scrum-oriented organisation tries to deliver functionality as fast as possible using already known techniques. Few such projects will allow for significant research dedicated to defining architecture and technical vision, or evaluate a variety of tools and technologies to find the best and most appropriate.

This approach can be compared to an athlete running a sprint and only focusing on a single point: the finish. This can work well for projects with low uncertainty staffed by teams with solid skill sets and experience that fit the needs of the task. Since CN is still new, though, many teams will not have that experience; going Cloud Native requires playing around with the new set of technologies and organisational practices until some facility is learned. In *this* race, the athlete needs to be looking up, down and to all sides, not just straight ahead.

Design thinking is the next-step process approach along the Maturity Matrix, and it might be a better fit, at least at the beginning of the CN initiative. Once the teams are confident with the new CN practices, they can switch back to Scrum if that helps to optimise the delivery process.

CONCLUSION TO INTRODUCTION

Identifying these common contexts helps us use Cloud Native design patterns effectively when solving the problems companies face when migrating to the cloud.

As we have seen, due to the complexity of distributed systems, full scale Cloud Native is difficult to implement. When coupled with the technical and cultural contexts most enterprises bring to the journey, the path ahead can seem formidable. Even with the help of an experienced Cloud Native consultant as guide.

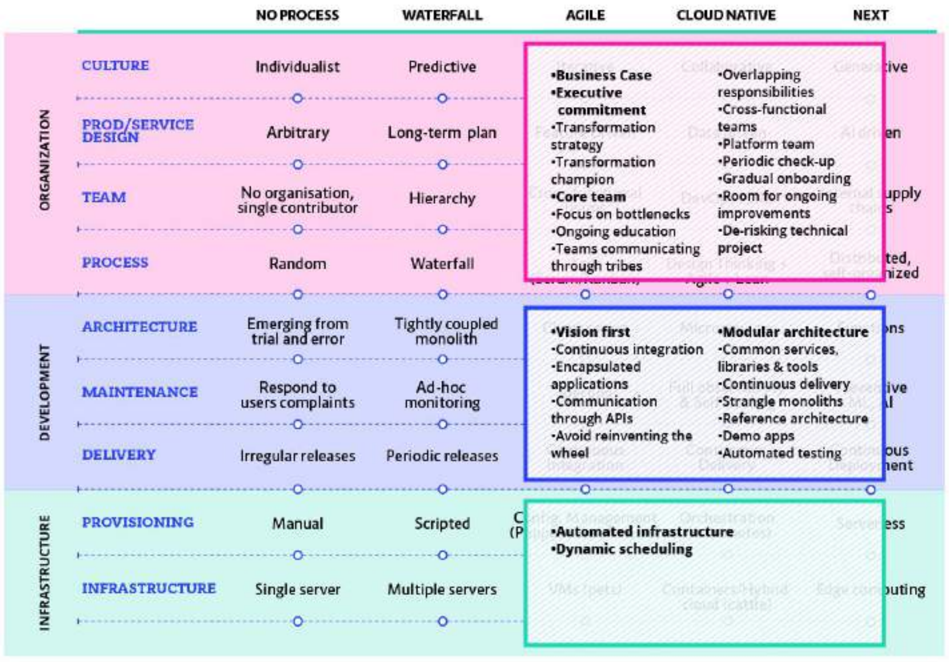
A means for smoothing that path is to develop a Cloud Native patterns language. A lingua franca allowing us to identify, teach, and implement context-specific best practices in this complex and evolving technology.

PATTERNS

The pattern language presented in this paper will eventually cover all the aspects listed in the Maturity Matrix and will go deeper into each subject. **The current list of patterns primarily focuses on the higher level patterns required in the beginning of a Cloud Native transformation**. Eventually, these will be expanded with more granular patterns. At this time the list includes:

- Business Case
- Executive commitment
- Core team
- Vision First
- Microservices Architecture
- Automated infrastructure
- Dynamic scheduling

Specific technical patterns are more common to find and will be published in future papers.



		NO PROCESS	WATERFALL	AGILE	CLOUD NATIVE	NEXT
ORGANIZATION	CULTURE	Individualist	Predictive			ive
	PROD/SERVICE DESIGN	Arbitrary	Long-term plan			en
	TEAM	No organisation, single contributor	Hierarchy			upply s
	PROCESS	Random	Waterfall			ted, nized
DEVELOPMENT	ARCHITECTURE	Emerging from trial and error	Tightly coupled monolith			ns
	MAINTENANCE	Respond to users complaints	Ad-hoc monitoring			ive l
	DELIVERY	Irregular releases	Periodic releases			ous ent
INFRASTRUCTURE	PROVISIONING	Manual	Scripted	C (P		ess
	INFRASTRUCTURE	Single server	Multiple servers			uting

- **Business Case**
- **Executive commitment**
- Transformation strategy
- Transformation champion
- **Core team**
- Focus on bottlenecks
- Ongoing education
- Teams communicating through tribes

- Overlapping responsibilities
- Cross-functional teams
- Platform team
- Periodic check-up
- Gradual onboarding
- Room for ongoing improvements
- De-risking technical project

- **Vision first**
- Continuous integration
- Encapsulated applications
- Communication through APIs
- Avoid reinventing the wheel

- **Modular architecture**
- Common services, libraries & tools
- Continuous delivery
- Strangle monoliths
- Reference architecture
- Demo apps
- Automated testing

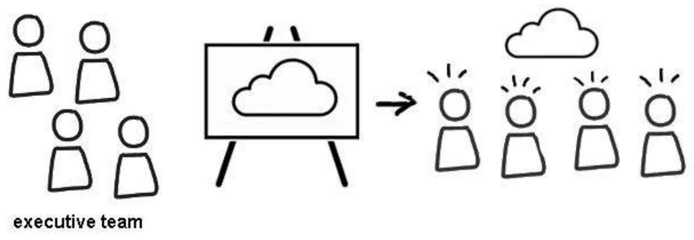
- **Automated infrastructure**
- **Dynamic scheduling**

Patterns marked in bold in this table are defined in fullest detail.
The rest will be expanded as full patterns in the future.

PATTERN LANGUAGE

Business Case

When an organisation's leadership does not fully comprehend the advantages that result from a Cloud Native migration, providing a strong Business Case will allow them to understand and support the project without hesitation.



A company is experiencing pressure from external advisors or internal tech teams to move to Cloud Native. The executive team is contemplating making the move to CN, but this is the first such transformation the company has undertaken and there is only partial understanding of the complexity of a CN migration and the benefits that will come from it.

In this context:

The benefits of the transformation are not clear to the executive team, so they may not support the initiative or even give it serious consideration.

- The traditional model is for organisations to be massively risk averse, to minimise uncertainty at all costs.
- Change-averse culture avoids new technologies or experimental approaches. Cloud Native architectures are conceptually different from traditional approaches, merging careful up-front planning with flexible and mutable, experimentation-based implementation.
- Tech teams are eager to get started with the transformation, even before business case is established

Therefore:

Create a formal business case to help educate the organisation's executive team, taking into account the benefits to be gained from Cloud Native.

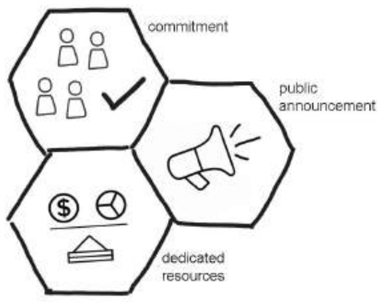
The business case needs to include key CN advantages, including acceleration of business velocity, scalability, potential cost savings, and enhanced recruitment and retention of tech staff.

Consequently:

The business case for a CN transformation is clear and company's decision makers have a clear understanding of the advantages CN confers and are ready to move forward.
They are prepared to allocate the necessary budget and resources that such a large project will require.

Executive Commitment

To ensure allocation of sufficient resources and reasonable delivery timeframes, large scale projects such as CN transformation require strong Executive Commitment.



You are working in an enterprise that is using Waterfall or Agile software development practices and there is a clear decision to adopt CN, with a Business Case supporting the transformation.

In this context:

Cloud Native transformation requires significant changes in all areas presented on the Maturity Matrix: infrastructure, development and organisation. These changes place large demands from the organisation in terms of budget and time allocation.

- Clients continue demanding fast delivery of new functionality leaving no slack for structural changes
- Executive performance is measured by P&L (profit and loss statement), that can reduce incentives to invest in long term structural improvement such as CN transformation

- Executives as well as the technical teams may not have the complete technical and organisational knowledge necessary for understanding the full scope of the CN transformation.
- Successful adoption of CN may significantly speed up the velocity of the feature development and increase the team satisfaction

Therefore:

Define Cloud Native transformation as a high priority strategic initiative with an explicit support from the executive management.
Such commitment from the management needs to include preparation of a Transformation Strategy (see related pattern) and the allocation of adequate resources and budget.
Public announcement of the CN transformation as a strategic initiative creates company-wide alignment and awareness, while also setting the expectation of collaboration from all departments within the organisation.

Consequently:

The company is aligned around common goals and everyone understand priorities for the transformation.
- All departments are working in collaboration to create a single strategy and unified vision, while avoiding independent silos that lead to inconsistent, or even conflicting, implementation.

Related Patterns:
Business Case, Ongoing Education, Vision First, Core Team, Transformation Strategy, Transformation Champion

Examples:
1. Bottom up transformation from from multiple sources in the organisation: Multiple teams starting to use public clouds, containers or schedulers independently and without any coordination with other departments within the organization. Typically by just using a personal or middle manager's credit card. This leads to variety of incompatible implementations that require very significant refactoring in order to work together which will typically fail to materialise, as some of systems will be already in production. Because, under pressure to deliver features, teams will have no time for refactoring or standardisation. This in turn will lead to a forest of smaller unrelated and

disorganized solutions, resulting in the waste of time and resources due to the inability to utilise economy of scale of large organisations.

In such situations, Executive Commitment is essential to provide an overarching vision and strategic goal for the teams to bring all independent solutions to a consistent and reusable state, while allocating necessary resources to achieve this.

2. Introduction of CN by ops department:
Operations department decides to introduce a dynamic scheduler such as Kubernetes and provides it to the development department. However, the needs of the development department have not been fully taken into account and so the implementation is heavily focused on the operational side. This typically creates significant overhead for developers and rarely has a good onboarding strategy. This leads to underutilisation of the platform by the developers and to shadow IT (alternative implementations of the platform) in its place.

   Making CN transformation a strategic initiative may help to tear down the walls between the departments and create a consistent platform that is both easy to use and easy to maintain while serving the needs of both sides.
3. Introduction of CN by dev department:
Similar to the previous example but coming from the development department. This leads to the creation of a platform without strong operational configuration. It is typically difficult to refactor the platform later on which creates significant overhead for support and stability.

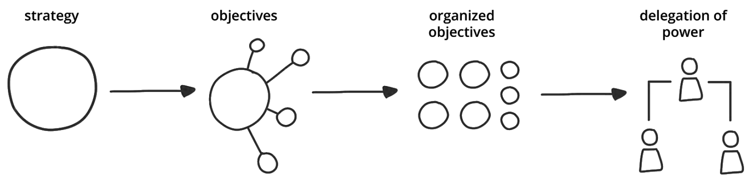
   Making CN transformation a strategic initiative may help to tear down the walls between the departments and create a consistent platform that is both easy to use and easy to maintain while serving the needs of both sides.
4. Demand for full transformation without sufficient resources and/or with unrealistic deadlines:
Company may be genuinely committed to the CN transformation, but the management team fails to fully appreciate the scope of transformation. The initiative is assigned as a small technical project that can be done by one or two engineers in the spare time then they are not busy with other tasks. Not enough budget is allocated for education, external help or appropriate tooling. This leads to the introduction of incomplete systems that is of use to few people in the organisation.

In such cases, executive commitment is required for the full scope, including executive education, technical experiments and other actions to make sure the the management team fully understands the job at hand and provides adequate support for a realistic execution plan.

Transformation Strategy

Once Executive Commitment is achieved, the management team can create a high level transformation plan and start delegating responsibilities to the teams. As the transformation moves ahead, the management team can monitor progress based on the objectives that have been defined.

Teams need to be independent enough to be able to interpret the objectives and translate them to actions within their own specific contexts.

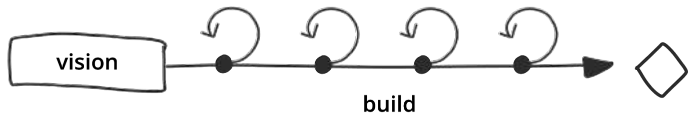


Transformation Champion

A person or small group of people leading and evangelising the transformation. The transformation champion person or team needs to understand both the subject and company objectives, be well connected within the organization, and highly motivated to promote the transformation.

Vision First

Defining a guiding vision as very first step helps set the right path through an uncertain environment.



The company needs to define a clear and achievable vision, which will later be translated into specific executable steps.

In this context:

The combination of limited experience and lack of extra time and flexibility for research leads to pursuing CN implementation using "well known ways".

- Without a overall consistent vision, different teams will make independent and, frequently, conflicting architectural decisions
- In many companies, Enterprise Architects are responsible for creating a detailed architecture. Many Enterprise Architects lack sufficient theoretical or practical experience in the Cloud Native approach.
- Agile methodologies, widely adopted in the contemporary business world, create pressure to produce results early and onboard teams to new systems very quickly.

Therefore:

You should define and visualize the architecture of the whole system upfront. It can either be requested from external sources or uncovered by series of small research and prototyping projects. It's important to keep the vision high level to allow freedom of choice during implementation, yet also detailed enough to provide clear guidance (which will help avoid common pitfalls).

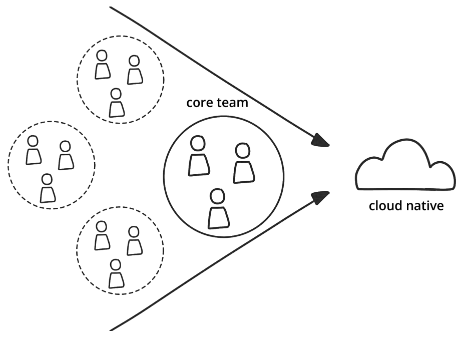
Consequently:

All teams have a clear guiding principle for the implementation phase. The teams can start producing the lower level architecture, and translate it to the backlogs of tasks. Therefore, Executive Commitment paired with leadership by the Transformation Champion are essential to have in place for successful vision creation.

Related Patterns:
Executive Commitment, Core Team, Transformation Champion

Core Team

A dedicated team of engineers and architects constantly diving deeper into technical challenges will reduce the risk of the transformation gaining experience that will help onboard the remaining teams more quickly and effectively.



With Vision First in place, the company is now allocating resources to the CN transformation and choosing the best teams for leading the initial stages.

In this context:

Existing teams working on building new features and ongoing improvements while still responsible for their original duties will have conflicting priorities. This can lead to insufficient resource allocation to the CN transformation project.

- Teams working both on both urgent and important tasks will tend to prioritise urgent tasks first, leading to deprioritization of important tasks such as CN transformation.
- CN technologies are new and complex. They require intense time investment for learning and experimentation.
- Some of the CN challenges are too difficult for one person to handle
- A team responsible and trusted for delivering a new solution will have full commitment to the solutions and later evangelise it across the organisation.

Therefore:

Create a Core Team of 5-8 engineers and architects to lead the transformation. Team responsibilities will include ownership of the technical vision and architecture, derisking the transformation by running a series of PoCs (Proof of Concepts), creation of MVP (Minimum Viable Product) and later on onboarding and guiding other teams. The team may continue improving the platform after the major parts of transformation are done.

Consequently:

The Core Team rapidly iterates through the most challenging parts of the transformation and paves the path for the rest of the teams in the company towards successful CN adoption.

The team is building knowledge and experience in the CN area, first using them to adjust the vision and the architecture of the applications as they go.

Later, the Core Team's first-hand understanding helps them to onboard other teams to the new way of working. The progress is visible and measurable.

Related Patterns:

Vision First, Gradual Onboarding, De-risking Technical Project, Reference Architecture, Demo Apps, Cross-functional Teams, Focus on Bottlenecks , Common Services, Libraries & Tools

Examples:

For the last 4 years, all of our CN transformation experiences included a Core Team. One organisation was HolidayCheck, an online travel site based in Switzerland.

When we came to HolidayCheck, the company had been working to introduce microservices, containers and other CN technologies for about two years. They had met with limited success, mainly due to lack of experience of working with these technologies while maintaining pressure on continually delivering new functionality.

The first and most important change we suggested was to introduce a Core Team of about 5-6 engineers and give them 3 months to experiment with the technologies and create the vision and architecture, implement a simple version of their platform and migrate one application to the new platform.

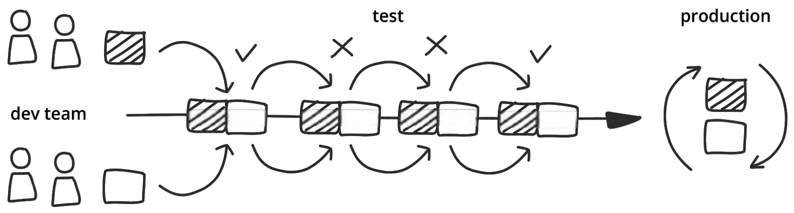
This change was successful. The team delivered the results almost within the deadlines, gaining useful knowledge in the process. After about 4 months they started onboarding other teams to the new platform. Following the successful onboarding, the team continued for several more months to finish building the platform and onboard remaining teams.

Once the platform was reasonably feature complete, the platform team fully functional and all the development teams on-boarded, the Core Team was not needed anymore: the transformation was now complete and the organisation was ready for the future. At that point all the Core Team members returned to their original teams and original tasks.

Ongoing Education

Cloud Native technologies are new and require significant learning effort. Prioritising ongoing education by encouraging engineers to learn through hands-on experimentation with the new technology helps them to onboard faster...and avoid some of the common mistakes related to new tools and platforms.

Continuous Integration



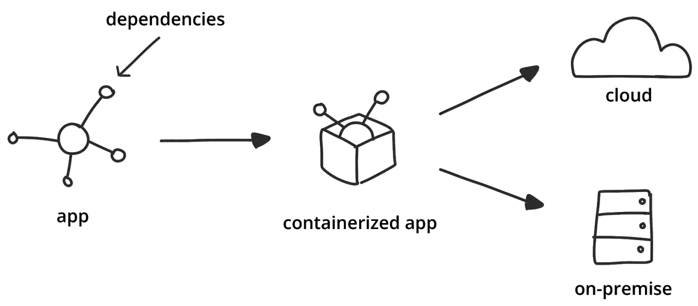
All teams are working on the same code base and integrating continuously, every day to reduce the integration burden. All the changes are thoroughly tested, fully and automatically ,on each submission.

These small but constant iterations reduce the cost and time of integration that leads to creating a higher quality of software and faster delivery of value to clients.

Encapsulated Applications

Cloud Native systems are responsive. They can change responsively to maintain their own stability. In a computer system that means recovering from failures like outages or crashes. This requires applications that can be rapidly restarted in the same or new locations, i.e. constantly deployed to a variety of platforms, quickly and in a reliable way.

Technologies such as software containers help to achieve this responsiveness by wrapping each application in a container that can run almost anywhere and which has very low overhead in terms of resources and startup time.



Communication Through APIs

Communication between independent components is done only through standard, stable and backwards compatible APIs.

APIs create strong boundaries between the components themselves and the teams building them. This way different teams can move at the pace that is comfortable for them while not slowing efforts for other teams, or creating the need for increased coordination.

New functionality can be added to a component and exposed through a new API without any effect to the rest of the system. Other components can then start using this functionality whenever they need it.

Automated Infrastructure

The absolute majority of operational tasks need to be automated. Automation reduces inter-team dependencies, which allows faster experimentation and leads in turn to higher development velocity.

Company is moving to CN and adopting CN patterns such as Microservices Architecture, Continuous Delivery and others. Teams are independant and require fast support services from the Platform Team. Most of the operational tasks are performed on demand by the Ops team.

In this context:

Manual or semi-automatic provisioning of infrastructure leads to dependencies between the teams and to long waiting times for results, hindering experimentation and slowing development velocity.

- Traditional operational teams don't have sufficient levels of automation and, due to high workload, no time to learn new technologies
- Public clouds provide full automation of infrastructure resources
- Manual requests and handover between development and operations teams is very slow
- Number of operations engineers in manual systems must scale up proportionally to growth in infrastructure demands
- Experimentation and research take longer and require more resources due to involvement of an already busy operations department.

Therefore:

Dedicate at least 50% of the Ops team's time on the automating the operational task and eliminate all manual infrastructure provisioning and maintenance tasks.
Infrastructure automation scripts need to be treated with equal importance as the rest of the company code base.
Automation needs to include compute, storage, networking, and other resources, patching and upgrading of operating systems, and deployment and maintenance of systems running on top of the infrastructure.
Full automation will allow the provisioning of exponentially more resources per member of operational staff.

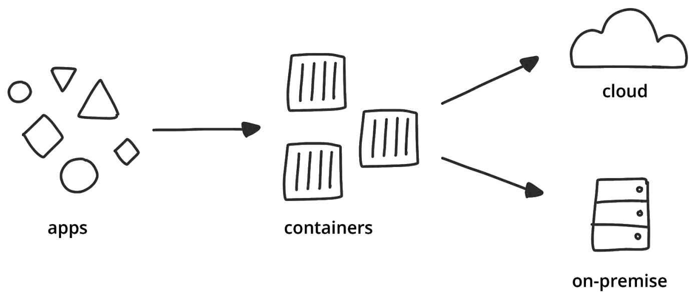
Consequently:

Developers spend less time waiting for infrastructure resources and are able to try out quick experiments, and to scale running systems rapidly and easily.
Ops team spending significantly lower amount of time on repetitive support tasks and investing more time and resources in ongoing improvement of the system.

Related Patterns:
Dynamic Scheduling, Version Control, Public Cloud, Private Cloud, Infrastructure Self Service

Dynamic Scheduling

Advanced technology companies deploy thousands of times a day to large number of development, testing and production environments; Dynamic Scheduling uses an orchestrator (Kubernetes) to organize the deployment and live management of applications.



Market demands that the company deliver value to clients in a very short period of time, such as hours or even minutes, therefore the company is moving to Microservices Architecture and using Continuous Delivery.
There are dozens of independent microservices and the development teams wants to deploy each one of them multiple times a day.

In this context:
Deployment of applications to static servers using manual or semi-automatic procedures cannot support the growing demands of the development teams to deploy each component separately on multiple environments once, or even more times, a day.

- Software systems become more distributed overall and are required to run on many platforms.
- Dynamic scheduling tools are becoming mature and available for general use
- Small parts of applications can fail at random times

Therefore:
All application scheduling needs to be done using dynamic schedulers in a fully automatic way.
Cross-functional teams need to understand how to use such tools effectively and they need to become part of the standard development process.

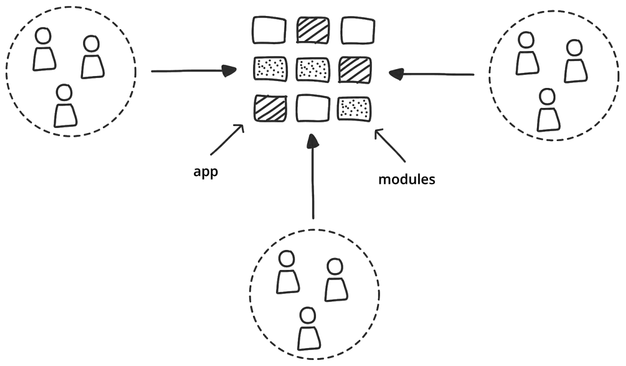
Dynamic scheduling also handles stability: restarting failing applications and autoscaling.

Consequently:
Developers build distributed systems and define how components will run and communicate with each other once they are deployed.
Applications can scale up and down, and non-functional parts can be restarted and healed automatically.

Related Patterns:
Continuous Integration, Continuous Delivery, Microservices Architecture, Cross-functional teams, Distributed Systems, Fast Experimentation Cycle, Encapsulated Applications

Microservices Architecture

To reduce the costs of coordination between teams delivering large monolithic applications, build the software as a series of microservices that are built, deployed and operated independently.



A company has decided to move to Cloud Native and is looking at the ways to increase the velocity of feature development and to optimise their utilization of cloud resources. The size of the development/engineering staff can range from a few tens, for a small to medium business, up to a few thousand for a large enterprise.

In this context:

Delivery of large monolithic applications developed by large teams require long and complex coordination and extensive testing, leading to longer TTM (Time to Market). Hardware utilisation by such applications is inefficient, which leads to waste of resources.

- People tend to delay painful moments; since integration and delivery are typically painful, their frequency tends to decrease as system longevity increases.
- Larger monolithic systems are increasingly more difficult to understand as they grow in size and complexity
- Monoliths are easier to work with than modular applications so long as they are small enough to be understood by each developer.
- Conway's law: architecture tends to resemble the organisational structure.

Therefore:

Split applications into smaller microservices that can be built, tested, deployed and run independently from other components.

- Independent components allow different teams to make progress at their own pace faster-moving teams are not held back by slower ones and to use the most appropriate tools for each situation.
- Independence and freedom of choice are achieved in a tradeoff with reduced standardisation and certain types of reusability.

Consequently:

New systems are created from a large number of small components with a complex web of connections.

- Small and independent teams work on separate modules and deliver them with only limited coordination across the teams.

Related Patterns:
Cross-functional teams, CI, CD, Common Services, Libraries & Tools, Communication Through API, Dynamic Scheduling,

Avoid Reinventing the Wheel

Off-the-shelf tools frequently lack one or more specific functionalities needed by the project at hand. At this point many development teams will consider building their own tool to create the perfect solution for their specific needs.

In almost all cases the better way is to stick with the existing tools to avoid costly creation and maintenance of a custom tool.

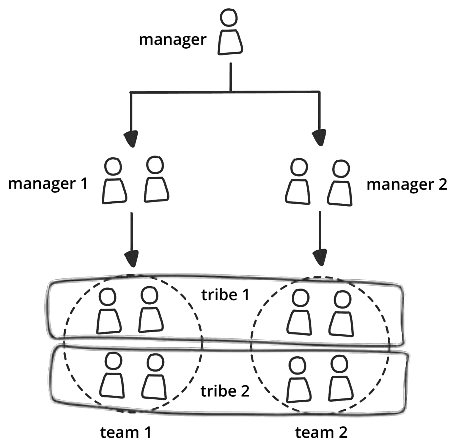
Common Services, Libraries & Tools

The Platform Team will only be responsible for a small set of core tools. Each team can introduce a new tool at their own risk and experimentally deploy it to the platform. Once the team has gained the experience of working with the tool and is confident the the tool is indeed needed, it can submit a request to the platform team to provide permanent support for it. After a handover and testing time at joint responsibility, the platform team will assume control and will be able to roll it out to the rest of the teams in the company.

Teams Communicating Through Tribes

Cloud Native technologies are distributed by their very nature. Under Conway's law, hierarchical organisational structure can still work for for administrative purposes, but it is insufficient for inter-team communication working on independent components of a distributed system.

Cross-team tribes can allow teams to efficiently exchange information on a variety of technical and other topics without losing the benefits of hierarchy required for compliance, resource allocation, etc.



Overlapping Responsibilities

It is not always clear who is responsible for each part of the system. In some cases, no one takes the responsibility for shared parts of the systems. At other times, teams might be arguing about who has control over different parts of the system.

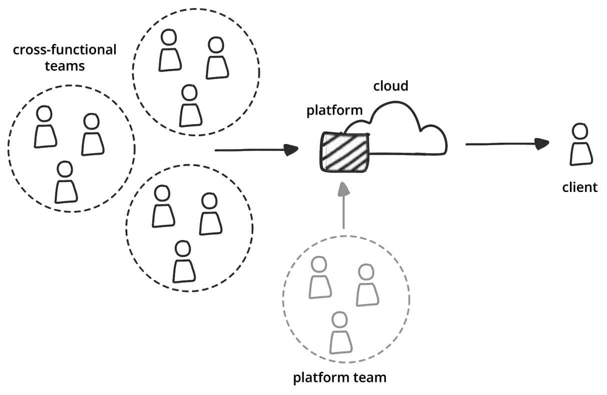
Joint responsibility can create both control and collaboration across the teams.

Family is a good example for overlapping responsibilities in real life. Who is responsible for washing the dishes or taking kids to school? Typically one member of the family has a stronger responsibility for each task, but still, everyone needs to do their chores.

It's typically unhealthy when there is a very strong and inflexible separation of duties in a family. Like families, teams benefit when responsibilities are shared.

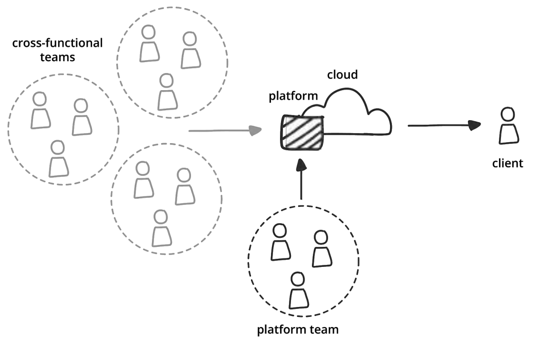
Cross-functional Teams

Teams working on Cloud Native applications (DevOps teams) need to be able to build, deploy and maintain distributed systems. Such teams need to be able to create microservices applications, package then in an encapsulated way (containers) and deploy them through CI/CD pipelines to dynamically scheduled clusters (Kubernetes).



Any platform used by such team needs to be fully automated and should not require any manual intervention at any stage (aside from unexpected problems and rare specialized maintenance tasks).

Platform Team



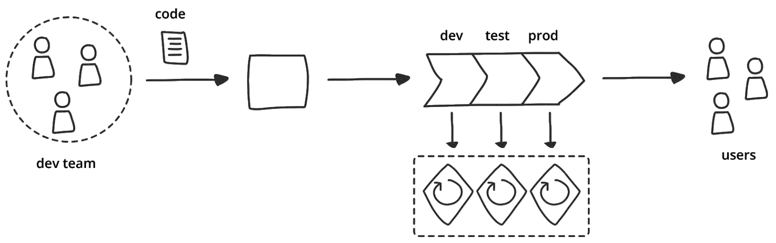
The Platform Team -- typically, the Site Reliability Engineering, or SRE, team -- is responsible for building and maintaining the platform that is used by the Cross-functional Teams. All common platform functionality needs to be programmable and accessible by the Cross-functional Teams.

Periodic Check-up

Typically, the organisation undertaking a Cloud Native migration defines the goal in the beginning of the transformation process and then moves fully into execution without occasionally stopping to assess progress. In many cases they fail to adjust course wherever the initial direction turns out to be incorrect.

Periodic check-ups can help to review the validity of the goals and explicitly change direction or confirm the current direction.

Continuous Delivery

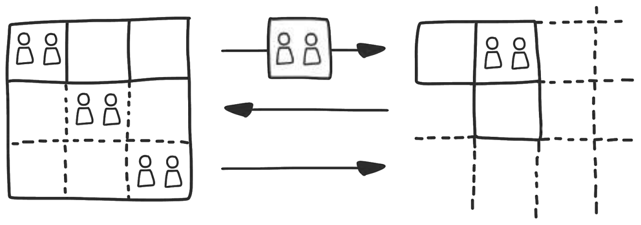


Given a growing number of independent components all delivered very frequently, teams must have fully automated and reliable delivery procedures.

Any delay for manual intervention or for quality issues will be compounded to considerable maintenance overhead for the platform and development teams due to the sheer number of moving pieces.

Continuous Delivery must be put in place before undertaking the move to Microservices Architecture.

Gradual Onboarding



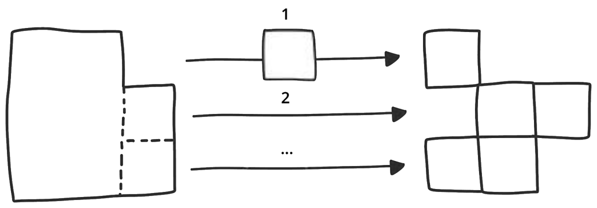
A newly deployed platform is typically not fully functional nor yet totally stable. There is not enough automation. The development teams have not gained experienced in using the platform.

As they supervise the simultaneous onboarding of a large number of teams, the platform team can become overloaded with support tasks for these teams. This will block further improvement of the platform. As a result, it will likely stagnate and fail to reach its full potential.

Instead, once the basic platform is set up, the platform team should onboard only 1-3 teams to start while continually improving the platform by fixing issues that emerge during the initial onboarding.

Continue in small team batches while continuously improving the platform.

Strangle Monoliths



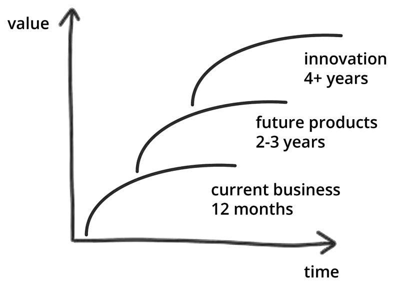
When not fully transitioned to the new modular architecture, the Cloud Native platform is not delivering its full value. The teams keep delivering slowly, held back as significant development continues in the monolithic portion of the application.

Create a simplified procedure to take small pieces of the monolith and rewrite them as separate modules. Reduce any new development of the monolith and allow only minimal maintenance. Plan to rewrite small pieces of the monolith all the time until it disappears completely.

Room for Ongoing Improvements

Each company needs to invest in future technologies and products. Without such investment, it will be difficult to change direction, adopt new technologies and, eventually, compete with other businesses that are capable of responding to customer's requests within just days or even hours.

McKinsey's Three Horizons of Growth is a framework for encouraging innovation and growth while ensuring current stability. The three horizons are H1: Maintain and Defend Core Business, or what your enterprise is doing right now. About 70-80% of organisational efforts should be directed here. H2: Nurture emerging business, focused on the next major product and consumes about 15-25% of effort. Finally, Innovation and next market should receive 5-15% of the overall attention for creating entirely new business elements. Essentially, research and development of new ideas that may be promising but unproven, and potentially unprofitable for a significant period of time. This would encompass things like research projects, pilot programs or entirely new revenue lines that require significant upfront investment.



It is important that all three horizons be in balance and receiving their proper share of attention and effort. Many established enterprises work only on H1 and forget about the future. Startups by definition are H2-H3.

SUMMARY

Enterprises that want to succeed in the digitized, Cloud-centric world will need to transform themselves to Cloud Native entities. Cloud Native can grant companies the ability to develop and deliver software faster, at greater scale, and at potentially lower cost. However, Cloud Native also comes with some unfortunate side effects most notably, the high complexity of distributed systems and the pain of cultural change. The shortage of knowledgeable and experienced Cloud Native developers creates additional difficulties for organisations seeking to transform themselves.

A Cloud Native Patterns Language addresses these difficulties by providing an effective way for developers, engineers and executives alike to identify “right” design patterns and the contexts necessary for their effective implementation. The Cloud Migration Maturity Matrix is a tool for assessing an organisation’s unique set of contexts for both their existing state and ultimate target objectives. Case studies from enterprises who have succeeded in transforming themselves into Cloud Native operations demonstrate the potent capabilities of Cloud Native patterns applied in proper context.

Acknowledgements

I would like to express my very great appreciation to:

1. Hans Wegener who acted as a shepherd during the writing process and helped me a lot with much great advice.
2. Anne Currie who helped me to clarify the ideas and make the sense out of vague concepts.
3. Michelle Gienow who helped with writing and editing of this paper.
4. Svitlana Chunyayeva and Rokas Raudonius from http://remembertoplay.co/ who helped visualising the patterns.
5. All https://container-solutions.com/ team who supported this project by sharing their real life experiences and by giving excellent feedback
6. Hugo Sereno Ferreira, Kyle Brown, Richard Gabriel and Joseph Yoder for reviewing the paper at PLoP 2018.

APPENDIX 1

Patterns thumbnails

ORGANIZATION PATTERNS		
Pattern Name	**Problem**	**Solutions**
Business Case	Company is contemplating making the move to CN, but there is only partial understanding of the complexity of a CN migration and the benefits that will come from it.	Create a formal business case to help educate the organisation's executive team, taking into account the benefits to be gained from Cloud Native.
Executive Commitment	Appropriate budget and attention are not allocated for the transformation in time which leads to partial transformation that is does not bring the expected benefits.	Define CN transformation as a strategic initiative with explicit support by executive management.
Transformation Strategy	Lack of coherent strategy leads to inability to evaluate progress by the execution teams	The management team needs to create a high level transformation plan and start delegating responsibilities to the teams.
Transformation champion	Transformation lacks a driving force, which leads to slow execution and low level of alignment	Appoint a person or small group of people to lead and evangelise the transformation.
Core Team	Transformation teams struggle to reach right level of motivation, capabilities, alignment or organisational support.	Create a single Core Team of 5-8 engineers and architects to lead the transformation.
Ongoing education	Cloud Native technologies are new and require significant learning effort.	Ongoing Education will allow engineers to onboard faster and avoid some of the common costly mistakes related to new tools and technologies.
Teams Communicating Through Tribes	Due to required amount of information exchange between independent teams and the complexity of the systems, teams find it difficult to communicate directly and efficiently.	Cross team tribes for variety of technical and other topics can allow teas to exchange information quickly and efficiently without losing the benefits of hierarchy required for compliance, resource allocation, etc.
Overlapping Responsibilities	Lack of clear responsibility leads to no attention to some parts of the system or to arguing about	Some parts of the system need to be under joint responsibility of multiple

	control over other parts of the system.	teams

ORGANIZATION PATTERNS Continued

Pattern Name	Problem	Solutions
Cross-functional Teams	Development teams are incapable of getting full benefits from Cloud Native systems due to lack of capabilities in some areas required to build such systems.	Create teams cross-functional teams with full Cloud Native capabilities
Platform Team	There is no consistent, fully automated and well supported platform that leads to constant instabilities and long waiting times for provisioning resources or making changes to the platform.	Create a platform team to build maintain a fully automated platform
Periodic Check-up	Cloud Native transformation goals and strategy and goals are defined in the beginning of the journey, but the teams are diverging which leads to inconsistent results.	Periodic check-ups can help to review the validity of the goals and explicitly change or re-confirm the current direction.
Gradual Onboarding	Due to initial instabilities of the platform and lack of knowledge and experience in the cross-functional teams, Platform support team can be easily overwhelmed by amount of support issues coming from number of teams onboarded too early and leading to stagnation in further improvement of the platform.	Once the basic platform is setup, onboard only 1-3 teams and continue improving the platform by fixing the issues discovered during the first onboarding. Continue onboarding in small batches while further improving the platform.
Room for Ongoing Improvements	Introduction of new technologies or new practices is difficult due to lack of time or adequate research capabilities.	Each company needs to invest into future technologies or products
De-risking technical project	Cloud Native transformation includes many risks related to new technologies and practices applied in variety of different situations. Risks are hidden and only discovered later on in the course of the transformation	Identify the riskiest and potentially most difficult issues visible in the beginning of the journey and run series of experiments to understand each challenge better.

DEVELOPMENT PATTERNS		
Pattern Name	**Problem**	**Solutions**
Vision First	Without a clear technical and organisational vision, teams are going in different directions leading to chaos.	Create a technical and organisational vision that is high level enough to allow teams freedom but also specific enough to give clear execution guidance. Make the vision available and clear to ALL
Continuous Integration	Manual build or test of software create significant delays in the delivery. Such delays may block the ability to deliver the changes continuously.	All teams working on the same code base and integrating continuously, every day to reduce the integration burden.
Encapsulated Applications	Development and deployment of application components to wide variety of target platforms is difficult due to variations in the environment.	All application components are packaged together with all needed dependencies and can be deployed anywhere.
Communication Through APIs	Communication through internal programming language function calls creates strong coupling of components and forces the teams to change and deliver them together which complicates and prolongs the delivery	Communication between independent components should be done only through standard, stable and backwards compatible APIs.
Microservices Architecture	Requirement to coordinate all development and operations teams before each release increases complexity cost and time of each release.	Split applications into smaller modules that can be built, tested, deployed and run independently from other components.
Avoid Reinventing the Wheel	Engineers tend to rebuild functionality available at the market. This is due to lack of awareness or the tuough the “they can do it better”.	Avoid rebuilding existing functionality, unless it is in the area of core business of the company.
Common Services, Libraries & Tools	Introduction of wide variety of new technology choices (tools, languages, processes, etc) for solving similar problems leads to duplication of work, which overloads the platform and cross-functional teams	Use only a small set of core tools. Introduction of new tools needs to go through predefined incubation process.
Continuous Delivery	Any delay for manual intervention or for quality issues will be compounded to significant maintenance overhead for the platform and	Deliver each application component independently and fully automatically, every day, or even more frequently.

	development teams due to the number of moving pieces.	

DEVELOPMENT PATTERNS Continued

Pattern Name	Problem	Solutions
Strangle Monoliths	The teams keep delivering slowly as significant development continues in the monolithic part of the application. Developers lose the motivation if they continue working on the old systems for too long.	Create a simplified procedures to take small pieces of the monolith and rewrite them as separate modules. Block any new development of the monolith and only allow minimal maintenance.
Reference Architecture	Every time starting a new component or a new application, development team is creating a new version of architecture which leads to inconsistency, difficulties in onboarding and higher maintenance load.	Create one or more, well documented, reference architectures to simplify and speedup creation of new projects
Demo Apps	Without simple code examples, developers are solving similar problem over and over again which leads to longer development process and many code variations.	Create simple Demo Application. Developers can copy-paste the code from the applications to reduce development time and increase consistency in code.
Automated Testing	Without fast and trusted test coverage, teams cannot deliver fast enough.	Create consistent and reliable test coverage using test pyramide principles.

INFRASTRUCTURE PATTERNS

Pattern Name	Problem	Solutions
Automated Infrastructure	Manual or semi-automatic provisioning of infrastructure creates delays for the development teams and block their progress.	Fully automate the infrastructure, including provisioning of compute, storage, networking, and other resources, patching and upgrading of operating system and deployment and maintenance of systems running on top of the infrastructure.
Dynamic Scheduling	Development teams cannot deploy application components at required frequency when the infrastructure is static and scheduling is	All application scheduling needs to be done using dynamic schedulers in a fully automatic way.

	inflexible.