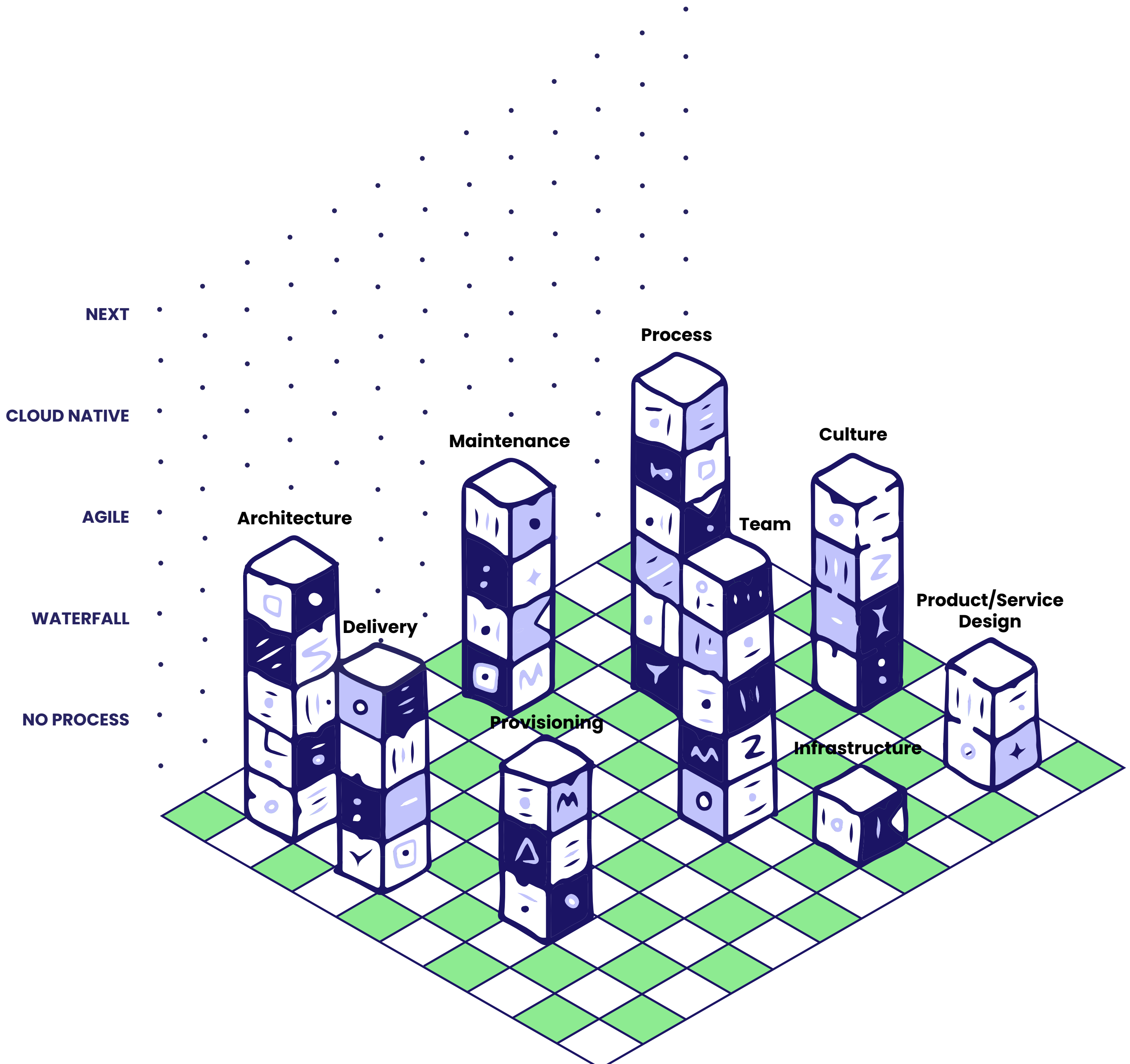
Container Solutions

What is the Cloud Native Maturity Matrix?



container-solutions.com

Container Solutions

Table of Contents

Introduction 2
Culture 8
Product/Service Design 11
Team 13
Process 15
Architecture 17
Maintenance 20
Delivery 23
Provisioning 27
Infrastructure 30
About Container Solutions 33
Want to Read More? *34*

Container Solutions

Introduction

Even if you are getting an inkling of WTF Cloud Native is, you might still be unsure about starting a journey toward it. Because Cloud Native is still new and mysterious. And if you're starting on a trip of any kind, it helps quell any anxieties you may feel if you know the answers to two questions:

- How far is our destination?
- Are we there yet?

At the core, Cloud Native is a method for optimising systems for the cloud. It is an approach to systems architecture that harnesses the cloud's most powerful advantages—flexible, on-demand infrastructure, and managed operational services—using Continuous Delivery.[1]

But it's a lot more than just some cutting-edge technologies that can make things go faster. To work properly—and not be an expensive, time-wasting boondoggle—Cloud Native tech needs to be accompanied by other changes. Changes in your organisation's culture, for instance. And in how you align IT and business goals strategically.

Most enterprises consider a Cloud Native transformation because they want to drastically speed up their ability to build, test, and deploy software, reducing that time from months to days or even hours. Beyond sheer velocity, the ability to continually deploy and update applications without ever disrupting users is the

[1] Continuous Delivery automates the delivery of small, iterative changes to run on cloud-based infrastructure. It provides an automated way to push code changes to teams working outside the production pipeline, performs any necessary service calls to web servers or databases, and executes procedures when applications are deployed.

ultimate goal of a cloud transformation. Companies that can't do this will quite simply get left behind.

If their software is developed on a Waterfall[2] or even Agile[3] model, those companies will find that customer expectations have shifted by the time their application finally deploys—moving them even further behind the competition. With Cloud Native delivery processes, however, those companies can more easily keep pace with rapidly shifting technology and customer demand.

Going Cloud Native is a complex process that few companies have deep experience in navigating. The tech is new. There's but a thin supply of engineers and developers who are fluent in it. And the path to transformation is different for nearly every organisation that sets out on the journey.

It makes sense that, in a young and quickly evolving sector, there were no maps to steer by.

So we made one.

Container Solutions have been guiding companies onto the cloud for six years now, carefully observing and analysing each experience. We took the lessons we learned to develop the Cloud Native Maturity Matrix. It's an assessment tool for helping map the path between where an organisation currently finds itself—and where it wants to be.

---

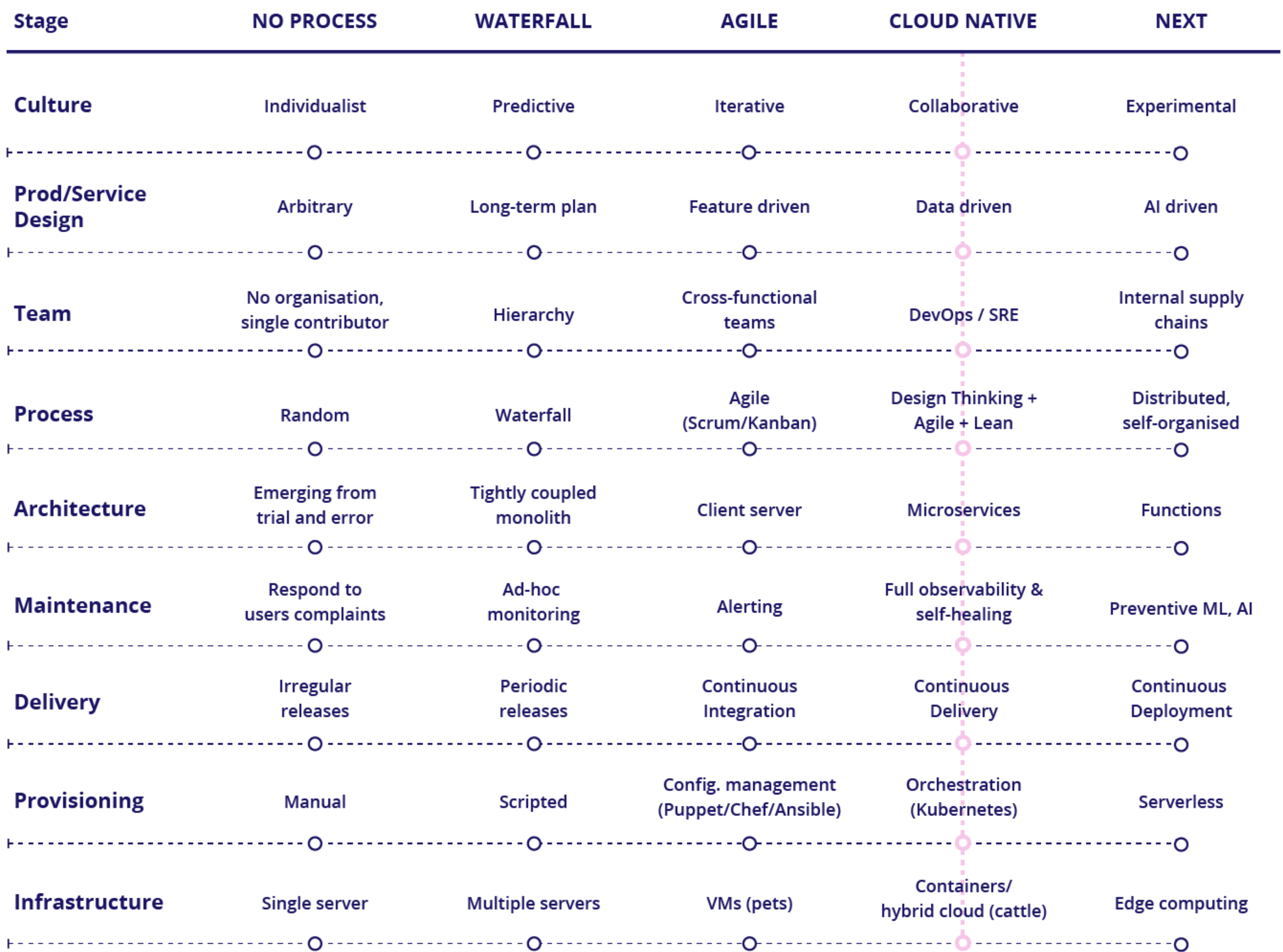
[2] A commonly used model of software development based on a logical progression of steps that form the software development life cycle. One follows after the other in strict order, much as a waterfall cascades down from top to bottom. This method is often associated with lengthy release cycles.

[3] Software development principles focusing on iterative delivery, frequent user feedback, and collaboration over complex processes. Scrum is the most well-known development method integrating the Agile principles. For example, sprints in Scrum implement the Agile principle 'deliver working software frequently'.

The Cloud Native Maturity Matrix uses interviews, workshops, and an assessment of the tech stack within an organisation to gather intel, which then is used to create a snapshot of that company along nine different axes. Some are technically oriented: infrastructure, maintenance and delivery. Others assess people-oriented aspects: management process, team structure, and internal culture.

We use the gathered information to define, analyse, and describe an organisation's current status in each of the nine categories. That status is literally plotted out on the matrix; at that point, the gap between the company's current state and a Cloud Native state is easy to see. This data—constantly re-assessed as things move forward—allows us to make intelligent choices and monitor progress.

In other words, the Cloud Native Maturity Matrix lets us create a custom map for each company's unique path to the cloud.



Stage	NO PROCESS	WATERFALL	AGILE	CLOUD NATIVE	NEXT
Culture	Individualist	Predictive	Iterative	Collaborative	Experimental
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-organised
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

Diagram 1: A blank Cloud Native Maturity Matrix, showing all nine categories.

Stage	NO PROCESS	WATERFALL	AGILE	CLOUD NATIVE	NEXT
Culture	Individualist	Predictive	Iterative	Collaborative	Experimental
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-organised
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
		Current situation		Goal	

Diagram 2: An example of a completed Cloud Native Maturity Matrix; the red area shows the gap between the organisation's current state in each category and Cloud Native status.

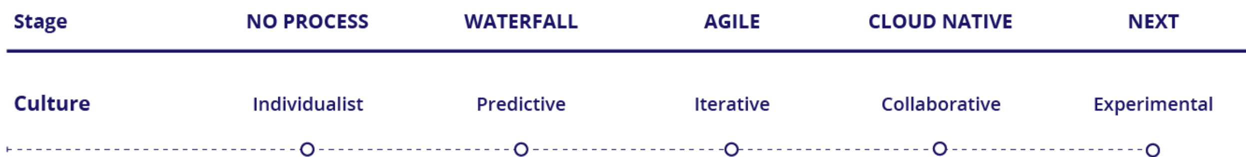
Container Solutions

This reference guide will help explain each category of the Cloud Native Maturity Matrix, and what Cloud Native status means in each area. We'll cover:

- Culture
- Product/Service design
- Team
- Process
- Architecture
- Maintenance
- Delivery
- Provisioning
- Infrastructure

Culture

Your Culture describes the way individuals in your organisation interact with one another.



No Process: Individualistic

In an individualistic organisation there is no approved way to interact with peers, supervisors, or subordinates. Instead, communications are rooted in personal preferences. The communications processes often change when the people in a team change. This is a common situation for startups, but becomes chaotic and unsustainable as you scale up.

Waterfall: Predictive

A predictive organisation embraces long-term planning and commits to deadlines. The goal of a predictive business is to deliver what was agreed, on time. Often the delivery will be large and complex. Delivering as fast as possible or exploring novel new ideas are not priorities; in fact, exploring new ideas is often actively discouraged.

In such an organisation you would expect to see large amounts of documentation; procedures for changes, improvements, and daily tasks; segregation of teams by specialisation; tools for every situation; and regular, lengthy planning meetings. Delivering an agreed specification on time is a difficult endeavour. Predictive companies need bureaucratic processes (for example, change control) and specialised team responsibilities (specific functions and technologies). This means

formal handovers between teams—for example, from development to test to operations. It also requires modest cooperation within teams and between teams coordinated by full-time project managers.

Predictive organisations tend to suppress novelty because it is essentially unpredictable. They value rule following, encourage permission-seeking, and punish deviation. All of these behaviours are logical, given the desire to deliver complex systems exactly as specified.

This culture is common in medium to large enterprises.

Agile: Iterative

An Agile organisation has similarities to a predictive one, but it chooses smaller and simpler goals, which it aims to deliver as fast as possible. Agile organisations tend to focus on the short term rather than following a long-term plan. Communication is often by short, daily meetings.

Culturally, Agile organisations prefer fast responses and quick fixes, which may lead to a 'hero culture' where individuals regularly display superhuman efforts to keep everything on track. They commonly use the Scrum project management methodology, with individual teams responsible for their part of the backlog. Inter-team communication is by Scrum masters and other coordinators. They typically have high cooperation *within* teams but modest cooperation *between* teams. Risks are usually shared within teams, but not between them. Like in a predictive organisation, Agile organisations normally have narrow responsibilities within a team, and narrow responsibilities for a team.

This culture is common throughout startups and enterprises of all sizes.

Cloud Native: Collaborative

A collaborative organisation tends to have big goals, but less specific ones than a predictive organisation. For instance, there may be a broad vision but without a detailed specification or a fixed delivery date. This culture embraces learning and consistent, continuous improvement over predictability.

Typically, this culture involves teams with full responsibility for their services, tools, and processes during the entire lifecycle—from design to deployment. High levels of collaboration exist within teams and between teams. There is often constant communication, using team chat tools like Slack. This culture rewards self-education, experimentation, and research. Results are coldly assessed based on field data.

A collaborative culture is increasingly being adopted in companies operating in areas of high uncertainty or fast change.

Next: Experimental

We predict the next type of organisation will be an experimental one. In an experimental culture, people within an organisation are encouraged to try new ideas on a small scale, learn from their failures, and scale up their successes.

Container Solutions

Product/Service Design

The Design category describes how decisions are made within your organisation about what product work to do next. What decides which products to develop, or which improvements or new features are tackled next?

Stage	NO PROCESS	WATERFALL	AGILE	CLOUD NATIVE	NEXT
Prod/Service Design	Arbitrary	Long-term plan	Feature driven	Data driven	AI driven

No Process: Arbitrary

An arbitrary design process is fad/wild-idea driven, somewhat random, and not deeply discussed. It is a common way to operate in startups where ideas usually come from the founders. On the upside, it can be highly creative. On the downside, it may result in partial features or an incoherent product.

Waterfall: Long-term plan

A long-term plan driven design process focuses on collating and assessing product-feature requests by customers, potential customers (via sales), users, or product managers. Individual features are then turned into team projects and multiple features are combined into large releases that happen every six to 12 months. This process is a common one for larger enterprises.

Agile: Feature-driven

A feature-driven design process speeds things up by allowing small new features to be selected with less planning. The aim is that these more modest features will be delivered to clients every few weeks or months in small batches. A feature-driven organisation focuses on fast change, often without an overarching long-term plan.

Cloud Native: Data-driven

In a data-driven design process, the final say on which features stay in a product is based on data collected from real users. Potential new features are chosen based on client requests or designs by product owners without a long selection process. They are rapidly prototyped and then potentially developed and delivered to users with copious monitoring and instrumentation. They are assessed against the previous features (better or worse?) based on A/B or multivariate testing. If the new feature performs better it stays; if worse, it is switched off or improved.

Next: Artificial Intelligence-driven

In the future, humans will be cut out of this process entirely! AI-driven systems will make evolutionary tweaks and test themselves with little developer interaction.

Container Solutions

Team

The Team axis describes how responsibilities, communication and collaboration works across teams in your organisation.

Stage	NO PROCESS	WATERFALL	AGILE	CLOUD NATIVE	NEXT
Team	No organisation, single contributor	Hierarchy	Cross-functional teams	DevOps / SRE	Internal supply chains

No Process: No organisation, single contributor

Here we find little structure, typically one or possibly a few independent contributors with no consistent management. This is most commonly found in small startups.

Waterfall: Hierarchy

A hierarchy organisation is organised via ranked positions within and between the teams. Decisions are made by managers and implementation is done by specialised teams (making it difficult to move individuals between teams). There will be separate teams of architects, designers, developers, testers, and operations engineers. Inter-team communication is generally through tools like Jira or via managers. Historically, this has been the most common structure in large organisations.

Agile: Cross-functional teams

In a cross-functional organisation there is less specialisation by teams, and more cross-capability within teams. For example, development teams will often include testing and planning capabilities. Scrum masters, product owners, etc. facilitate

communication between teams. However, a hierarchy remains outside (rather than within) teams.

Cloud Native: DevOps/SRE

A DevOps team is a development team capable of designing and building applications as part of a distributed system, and also operating the production platform/tools. Each team has full responsibility for delivering microservices[4] and supporting them. DevOps teams include planning, architecture, testing, dev, and operational capabilities.

However, often there's some separation of tasks. For example, it is common to see a platform DevOps team in charge of building the Cloud Native platform, while Site Reliability Engineers (SRE) or 1st level support teams respond to alerts. However, there is considerable collaboration between those teams and individuals can easily move between them.

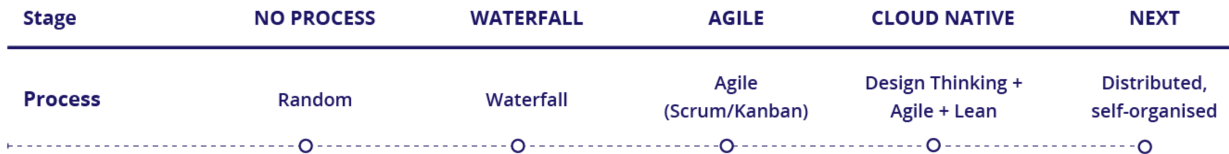
Next: Internal supply chains

In an internal supply chain organisation each service is a separate product, with full tech and business generation responsibilities in the teams—much as many e-commerce teams have been managed for a decade.

[4] An approach to application development in which a large application is built as a suite of modular components or services. Each service runs a unique process and usually manages its own database. A service can generate alerts, log data, support UIs and authentication, and perform various other tasks. Because microservices enable each component to be isolated, rebuilt, redeployed, and managed independently, development teams can take a more decentralized (nonhierarchical) approach to building software.

Process

Process describes how your organisation executes its work.



Stage	NO PROCESS	WATERFALL	AGILE	CLOUD NATIVE	NEXT
Process	Random	Waterfall	Agile (Scrum/Kanban)	Design Thinking + Agile + Lean	Distributed, self-organised

No Process: Random

In a random organisation, there is no change-management process, just random changes made at will. There is often no consistent versioning. This is common in many small companies with only a couple of engineers.

Waterfall: Waterfall

In a Waterfall organisation, the product-development process is tightly controlled through up-front planning and change-management processes. A sequential process is followed by planning, execution, testing, and finally delivery. There is usually an integration stage before delivery, where work from different streams is combined.

The process is run by managers and any and every handover is well documented and requires forms and procedures.

Agile: Agile (Scrum/Kanban)

In an Agile organisation, product development is run in sprints using an Agile technique such as Scrum or Kanban. Documentation is limited (the product is the documentation) and teams are heavily involved in their own management through daily consultation. There is usually considerable pressure to deliver fast, and no

defined provision for experiments or research. Only limited changes, if any, are allowed during sprints to protect the delivery deadlines.

Cloud Native: Design Thinking + Agile + Lean

In a Design Thinking organisation, Design Thinking and other research and experimentation techniques are used for de-risking large and complex projects. Many proofs of concept (PoCs), or small experiments, are developed to compare options. Kanban is often then used to clarify the project further; finally, Scrum is used once the project is well understood by the entire team.

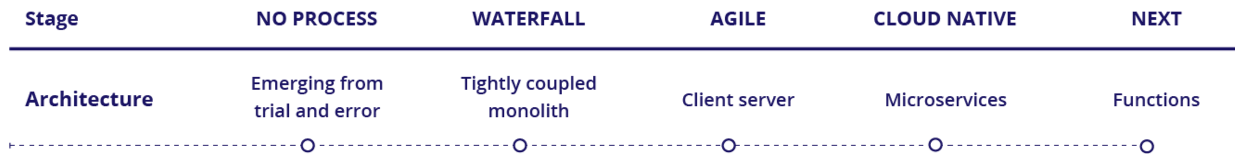
This relatively new process can be used in situations of high uncertainty or where the technology is changing rapidly.

Next: Distributed, self-organised

In the future, self-organised systems will be highly experimental, with less up- front design. Individuals or small teams will generate ideas that are iterated and improved on in the field automatically by the platform.

Architecture

Architecture describes the overall structure of your technology system.



Stage	NO PROCESS	WATERFALL	AGILE	CLOUD NATIVE	NEXT
Architecture	Emerging from trial and error	Tightly coupled monolith	Client server	Microservices	Functions

No Process: Emerging from trial and error

In an architecture described as emerging from trial and error, there are no clear architectural principles or practices. Developers just write code independently and all system-level communication is ad hoc. Integrations between components tend to be poorly documented, unclear, and hard to extend and maintain.

Waterfall: Tightly coupled monolith

A tightly coupled monolith is an architectural model where the entire codebase is built as one to five modules, with many developers working on the same components. A layered architecture (database, business logic, presentation, etc.) is common. Although interfaces have been defined, changes in one part often require changes in other parts because, typically, the code is divided into components with very strong coupling.

Delivery is done in a coordinated way, all together. Typically, the monolith is written in a single programming language with strong standardisation on tooling. The application is usually vertically scalable (you can support more users by adding more resources on a single server).

The design and maintenance of the monolith is usually led by a system architect or her team—many of whom are not hands-on developers. Unfortunately, there are

few developers or architects who can hold the entire system in their heads. Most people don't understand the full complexity of the app and that the impact of a single bug may be unpredictable and create domino effects that can destabilise the system overall.

Agile: Client server

A client server architecture is the most basic form of distributed system. It is designed to handle a system of multiple components communicating through networks that might be slow or unreliable. Each component is often similar to a monolith with a layered architecture. Each service can be clustered (which enables targeted horizontal scaling and resilience).

Like a monolith, in a client-server architecture multiple teams work on services at once and all services need to be deployed together. However, because the network-induced separation provides a degree of decoupling, the system is usually possible to develop while working in parallel to some degree (one group handles the client part, one the server).

Cloud Native: Microservices

A microservices architecture is highly distributed. It comprises a large number (usually more than 10) of independent services that communicate only via well-defined, versioned APIs. Often, each microservice is developed and maintained by one team. Each microservice can be deployed independently and each has a separate code repository. Hence, each microservice team can work and deploy in a highly parallel fashion, using their own preferred languages and operational tools and datastores (such as databases or queues).

Operationally, it is common to manage microservice deployment in a fully automated way. Because the system is distributed and components are decoupled

not only from each other but from other copies of themselves, is it easy to scale the system up by deploying more copies of each service.

Next: Functions

A functions (aka serverless) architecture is one where no infrastructure needs to be provisioned. Each piece of business logic is in a separate function, which is operated by a fully managed Function-as-a-Service, such as AWS's Lambda, Microsoft's Azure Functions, or Google's Cloud Functions.

No operations tasks—such as up-front provisioning, scaling or patching—are required. There is a pay-as-you-go/pay-per-invocation model.

Container Solutions

Maintenance

The Maintenance category describes how software is deployed and then run in a production environment in your organisation.

Stage	NO PROCESS	WATERFALL	AGILE	CLOUD NATIVE	NEXT
Maintenance	Respond to users complaints	Ad-hoc monitoring	Alerting	Full observability & self-healing	Preventive ML, AI

No Process: Respond to users' complaints

A response to users' complaints process is one in which the development and operations teams are alerted to most problems only when users encounter them. There is insufficient monitoring to flag issues in advance and allow engineers to fix them before the majority of users encounter them. System downtime may only be discovered by clients, or randomly. There is no alerting.

For diagnosing issues, administrators usually need to login to servers and view each tool/app log separately. As a result, multiple individuals need security access to production. When fixes to systems are applied, a manual upgrade procedure is followed.

This is a common situation in startups or small enterprises but it has significant security, reliability, and resilience issues, as well as single points of failure (often individual engineers).

Waterfall: Ad-hoc monitoring

An ad-hoc monitoring process consists of partial monitoring of system infrastructure and apps. This includes constant monitoring and alerting on basic, fundamental downtime events such as the main server becoming unresponsive.

Live problems are generally handled by the operations team and only they have access to production. Still, there's usually no central access to logs and engineers must login to individual servers for diagnosis, maintenance operations, and troubleshooting. Update processes may still be manual but formal runbooks (documentation) and checklists exist for performing update procedures.

This is a very common situation in enterprises. It still has security, reliability, and resilience issues.

Agile: Alerting

An alerting process involves pre-configured alerts on a variety of live system events. There is typically some log collection in a central location but most of the logs are located in separate places.

Operations teams normally respond to these alerts. Operations will escalate to Developers if they can't resolve the issue. Operations engineers still need to be able to login to individual servers. Update processes, however, may be partially or fully scripted.

Cloud Native: Full observability and self healing

In full observability and self healing, the system is highly monitored. Many issue responses happen automatically; for example, system health checks may trigger automatic restarts if failure is detected. Alternatively, the system may gradually degrade its own service to keep itself alive if, for example, resource shortages such as low disk space are detected. Status dashboards are often accessible to everyone in the organisation so that they can check the availability of the services.

Operations (sometimes called 'platform') engineers respond to infrastructure and platform issues that are not handled automatically. Live application issues are handled by Development teams or SRE teams.

Logs are all collected in a single place. This often includes distributed tracing output. Operations engineers, developers, and SREs all have access to the logging location. They no longer have (or need) security access to production servers.

All update processes are fully automated and do not require access by individual engineers to individual servers.

Next: Preventative machine learning, artificial intelligence

In the next generation of systems, machine learning and artificial intelligence will handle operational and maintenance processes. Systems learn on their own how to prevent failures by, for instance, automatically scaling up capacity.

Humans are defining the starting point for machines to learn and constantly improve. Self healing is the optimal way for systems to be operated and maintained. It is faster, more secure, and more reliable.

Delivery

The Delivery process describes how and when software from your development teams gets to run in your live (production) environment.

Stage	NO PROCESS	WATERFALL	AGILE	CLOUD NATIVE	NEXT
Delivery	Irregular releases	Periodic releases	Continuous Integration	Continuous Delivery	Continuous Deployment

No Process: Irregular Releases

In many small organisations, irregular software releases (new functions or fixes) are delivered into production at random times based on IT or management decisions about the urgency of the change. For highly urgent issues, like fixes for production problems, changes are delivered by developers directly to production ASAP.

This is a common situation for startups and small enterprises.

Waterfall: Periodic releases

Many organisations have periodic scheduled releases—for example, every six months. The contents of these, usually infrequent, releases becomes extremely important and are the result of long planning sessions. Extensive architectural documents for each release are produced by enterprise architects and requirement documents by business analysts. No coding is done before the full architecture is ready. Once the release contents are agreed, any change is subject to a Change Approval Board. A key driver behind infrequent releases is the need to perform expensive manual testing of each release prior to deployment.

Highly sequential processes are followed for each release:

1. System and software requirements are captured in a product requirements document.
2. Analysis is performed, resulting in documented models, schema, and business rules.
3. Design of the software architecture is completed and documented.
4. Coding is done: the development, proving, and integration of software (i.e. merging the work done by different teams).
5. Testing of that integrated new code is performed, including manual tests.
6. The installation and migration of the software is completed by the operations team.

After the release, the Operations teams support and maintain the complete system.

Agile: Continuous Integration

Continuous Integration describes an organisation that ensures new functionality is ready to be released at will—without needing to follow a strict release schedule (although a formal release schedule may still be followed). It often results in more frequent releases of new code to production.

A tech organisation using Continuous Integration typically:

- Has a single codebase (aka a source repository) that all developers add their code to. This ensures that merging and integration happen constantly rather than occasionally. That tends to make merging much easier.
- Has a fully automated build process that turns new code into runnable applications.

- As part of the build, includes automated testing of all code. That forces developers to fix bugs as they go along, which is easier than fixing them late in the process.
- Requires developers to add their new code to the single repository every day, which forces them to merge and fix bugs incrementally as they go along.
- Has a way to deploy code to test or production hardware in an automated fashion.

Cloud Native: Continuous Delivery

Continuous Delivery describes an organisation that ensures new functionality is released to production at high frequency (often several times per day). That does not mean the new functionality is exposed to all users immediately. It might be temporarily hidden or reserved for a subset of experimental or preview users.

A tech organisation using Continuous Delivery typically:

- Has a so-called 'deployment pipeline' where new code from developers is automatically moved through build and test phases.
- New code is accepted (or rejected) for deployment automatically.
- Thorough testing of functionality, integration, load, and performance happens automatically.
- Once a developer has put their code into the pipeline, they cannot manually change it.
- Individual engineers do not have permission to change the production (live) servers.

Companies using Continuous Delivery usually display continuous systems improvements. They also run tests on their production systems using methods such as chaos engineering (a way of forcing outages to occur on production

systems to ensure those systems recover automatically) or live testing for subsets of users (A/B testing).

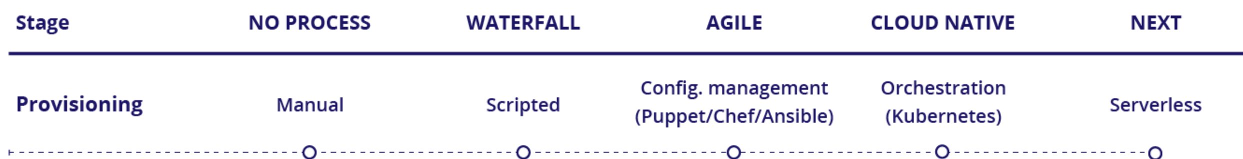
Next: Continuous Deployment

The next evolution of delivery is Continuous Deployment. In an organisation using this process, we see fully automatic deployment to production with no approval process—just a continuous flow of changes to customers. The system will automatically roll back (uninstall new changes) if certain key metrics—such as, say, user conversion—take a hit.

Container Solutions

Provisioning

The Provisioning process describes how you create or update your systems in your live production environment.



No Process: Manual

In a manual system, a developer (who is also your operations engineer) logs in to a server and starts apps manually or with rudimentary scripting. Servers are accessed using primitive file transfer mechanisms like FTP.

This is a common situation in startups. It is slow, labour-intensive, insecure, and doesn't scale.

Waterfall: Scripted

In a scripted system, developers build an app and hand it over to the Operations team to deploy it. The Ops team will have a scripted mechanism for copying the application and all its dependencies onto a machine to run. They will also have a scripted mechanism for configuring that machine or they may have pre-configured virtual machines (VMs).

In this case, because the Development team 'throws their app over the wall' to Operations, there is a risk that the developers built and tested their app using different tools, versions, or environments to those used by the Ops team. This can cause an application that worked fine for the Dev team to fail to work when Operations puts it on its test or live servers. This introduces confusion when issues

are subsequently seen: is there a bug in the app delivered by Dev or is it an issue in the production environment?

Agile: Configuration Management (Puppet/Chef/Ansible)

In a system with Configuration Management, applications are developed to run on specific hardware or virtual machines. Commercially available or open source configuration tools like Puppet, Chef, or Ansible allow operations engineers to create standardised scripts, which are run to ensure a production system is configured exactly as required for the application provided by Development. This can be done at will (in other words, fast) but there is limited automation (mostly a human presses a button to run the scripts).

Developers often deploy on their local test environments with different, simpler tooling. Therefore, mismatches can still occur between developer environments and production ones, which may cause issues with the live system. However, this is less common and faster to resolve than with more ad-hoc scripting.

Cloud Native: Orchestration (Kubernetes)

In a system with Orchestration, applications in production are managed by a combination of containerisation (a type of packaging that guarantees applications are delivered from development with all their local operational dependencies included) and a commercially available or open-source orchestrator such as Kubernetes.

The risk of a mismatch between development and live environments is reduced or eliminated by delivering applications from Dev to Ops in containers along with the app's dependencies. The Ops team then configures Kubernetes to support the new application by describing the final system they want to produce in production. This is called declarative configuration.

The resulting system is highly resilient, automated, and abstracted. Neither engineers nor the apps themselves need to be aware of hardware specifics. Everything is automatic. Detailed decision making about where and when applications will be deployed is made by the orchestrator itself, not a human.

Next: Serverless

It is now becoming more common for companies to be serverless—to give up Ops or even DevOps and allow all hardware maintenance and configuration to be done in a fully automated way by what is usually a cloud platform.

Code is packaged by developers, submitted to the serverless service, and can potentially be distributed and executed on many different platforms. The same function can run for testing or live. Inputs, outputs, and dependencies are tightly specified and standardised.

Container Solutions

Infrastructure

Infrastructure describes the physical servers or instances that your production environment consists of: what they are, where they are, and how they are managed.

Stage	NO PROCESS	WATERFALL	AGILE	CLOUD NATIVE	NEXT
Infrastructure	Single server	Multiple servers	VMs (pets)	Containers/ hybrid cloud (cattle)	Edge computing

No Process: Single server

In a single server environment you run all of production on a single physical machine. This may be an old desktop sitting under a desk in the office. You have no failover servers (resilience) and you deploy to your server using copy-and-paste file transfers. You probably have some rudimentary documents to describe the setup.

Waterfall: Multiple servers

A multiple servers (physical) infrastructure will handle a moderately complex application. You may have some redundancy (if one machine fails, another will take over) and you can have a sophisticated system of multiple interacting applications—for example, front ends and a clustered database. This is probably all sitting in a simple, co-located data centre.

Your Operations team may use manual problem solving and it might take days or weeks to provision new infrastructure because it's hard to get more rackspace! Compute, storage, networking, and security are usually managed separately and require separate requests to Ops. New infrastructure is ordered through a ticketing system and provisioned by Ops.

Container Solutions

Agile: Virtual machines (pets)

A virtual machines (pets) based environment is similar to a multiple servers environment in that you have a set of machines and manual server setup. However, this is made easier by using standardised virtual machine images. You use virtualisation software, such as VMWare, to help manage your virtual machine instances. You get better resource utilisation (and therefore an effectively larger system for your money) by running multiple VM instances on each physical server.

Your operations team uses manual or semi-automated provisioning of new infrastructure resources. Your VMs are 'mutable' —meaning, engineers can log on to them and change them by, for example, installing new software or fixes. Each machine is maintained separately and it would be painful if one died (hence, 'pets'). It will generally take hours or days to provision new infrastructure, mainly due to handovers between Dev and Ops teams.

Cloud Native: Containers/Hybrid cloud (cattle)

In a containers/hybrid cloud (cattle) environment, individual machines don't matter. Ops or DevOps don't directly provision infrastructure resources like VMs, they are only accessed through automated processes exposed through APIs.

It takes minutes or seconds to provision new infrastructure, always through APIs. Containers are often used for application packaging, which makes it easier to run those applications on multiple different 'hybrid' cloud environments (on prem or public). There is usually full automation of environment creation and maintenance. Under normal conditions, your engineers have no manual access to physical infrastructure. If any piece of infrastructure fails you don't care—it can easily be recreated (hence, 'cattle').

Next: Edge computing

Container Solutions

The next evolution for infrastructure is edge computing. Compute loads are run on edge devices—i.e. outside of your normal data centre. Edge computing returns results fast and works well where, for example, enough data is available locally and network connections to central data centre locations may be unreliable.

Container Solutions

About Container Solutions

Container Solutions is a professional services firm that specialises in Cloud Native computing.

Our company prides itself on helping enterprises migrate to Cloud Native in a way that is sustainable, integrated with business needs, and ready to scale. Our proven, four-part method, known as Think Design Build Run, helps companies increase independence, take control, and reduce risk throughout a Cloud Native transformation.

The process is stepwise to minimise risk, but delivers value quickly. In our Think phase, we listen carefully to people throughout a company, from the boardroom on down, to alleviate challenges and pain points, and formulate strategy. In the Design phase, we conduct small experiments to eliminate wrong choices and help organisations select the best path forward, regardless of vendor. In the Build phase, we collaborate with our clients' engineers to create a Cloud Native system aimed at delivering software faster and easier. In the Run phase, we train our customers' engineers to maintain their new system themselves—though we also offer partial or full, 24/7 operations support if they prefer.

Container Solutions is one of only a handful of companies in the world that are both part of the Kubernetes Training Partner (KTP) programme and a Kubernetes Certified Service Provider (KCSP). Membership to both programmes is based on real-life, customer experience. When companies like Google, Atos, Shell, and Adidas need help with Cloud Native, they turn to Container Solutions. We are a remote-first company that operates globally, with offices in the Netherlands, the United Kingdom, Germany, and Canada.

Container Solutions

Want to Read More?

Check out these Container Solutions publications:

The Resilient Company: Cloud Native Patterns to Help Navigate a Crisis

by Pini Reznik

A Pattern Language for Strategy

by Jamie Dobson and Pini Reznik

WTF are Microservices for Managers?

by Riccardo Cefala