THE CLOUD NATIVE MATURITY MATRIX

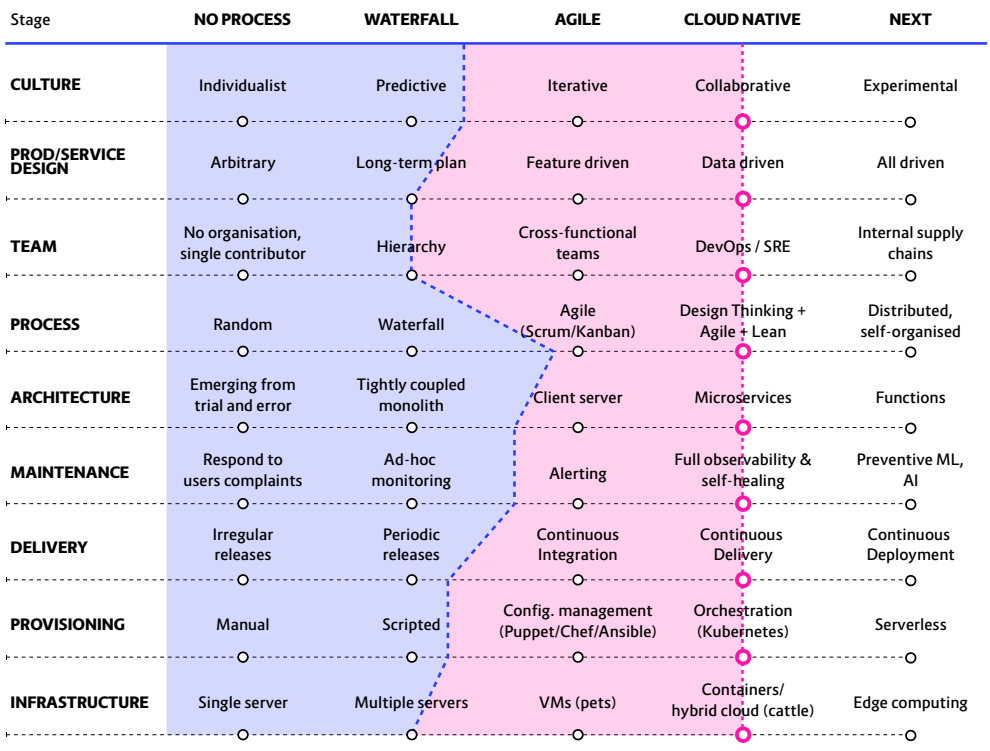
Questionnaire

INTRODUCTION

Cloud Native is a method for optimising systems for the cloud. It harnesses the cloud's most powerful advantages—flexible, on-demand infrastructure and managed operational services—and pairs them with continuous delivery, containers, microservices, and other cloud-optimised technologies. When done right, it allows you to drastically speed up building, testing, and deploying software—and doing so easily and often without ever disrupting user experience.

Migrating an organisation's existing systems onto the cloud is a complex process, not to be undertaken in haste. Some companies believe they can simply 'lift and shift' their current operations onto the cloud. They quickly find out, however, that this is not true. A successful cloud migration requires a complete transformation of an organisation's technology as well as human-centred aspects like culture and leadership. Cloud Native is very new, and few companies have deep experience in navigating the architecture's complexities.

THE MATURITY MATRIX



A sample Maturity Matrix outcome graph from a real-world enterprise assessment. It shows both the company's current status and the progression points necessary for transforming to Cloud Native.

Through the past five years spent guiding companies onto the cloud, Container Solutions have been carefully observing and analysing each experience. From these lessons we developed Container Solutions Cloud Native Maturity Matrix, an assessment tool for mapping an effective transformation path. The Maturity Matrix investigates nine different areas of transformation. Some are technically oriented, others assess cultural aspects of your company. We use the gathered information to define, analyse and describe your current organisational status—and then create your own custom roadmap for an effective cloud migration.

This questionnaire will guide you through a Maturity Matrix self-assessment. This is a lighter weight version of the same tool CS engineers use when performing our in-depth Cloud Native Readiness Assessment, a multi-day dive into a company's current systems, processes, and culture. This basic version provides an insightful snapshot of where you are right now, so we can map the fastest—and lowest risk—flight path for your company's unique Cloud Native transformation.

If any of the concepts you encounter are unfamiliar or confusing, please refer to the Glossary of Cloud Native Terms at the back of this booklet.

INSTRUCTIONS

How do we assess current organisational status?

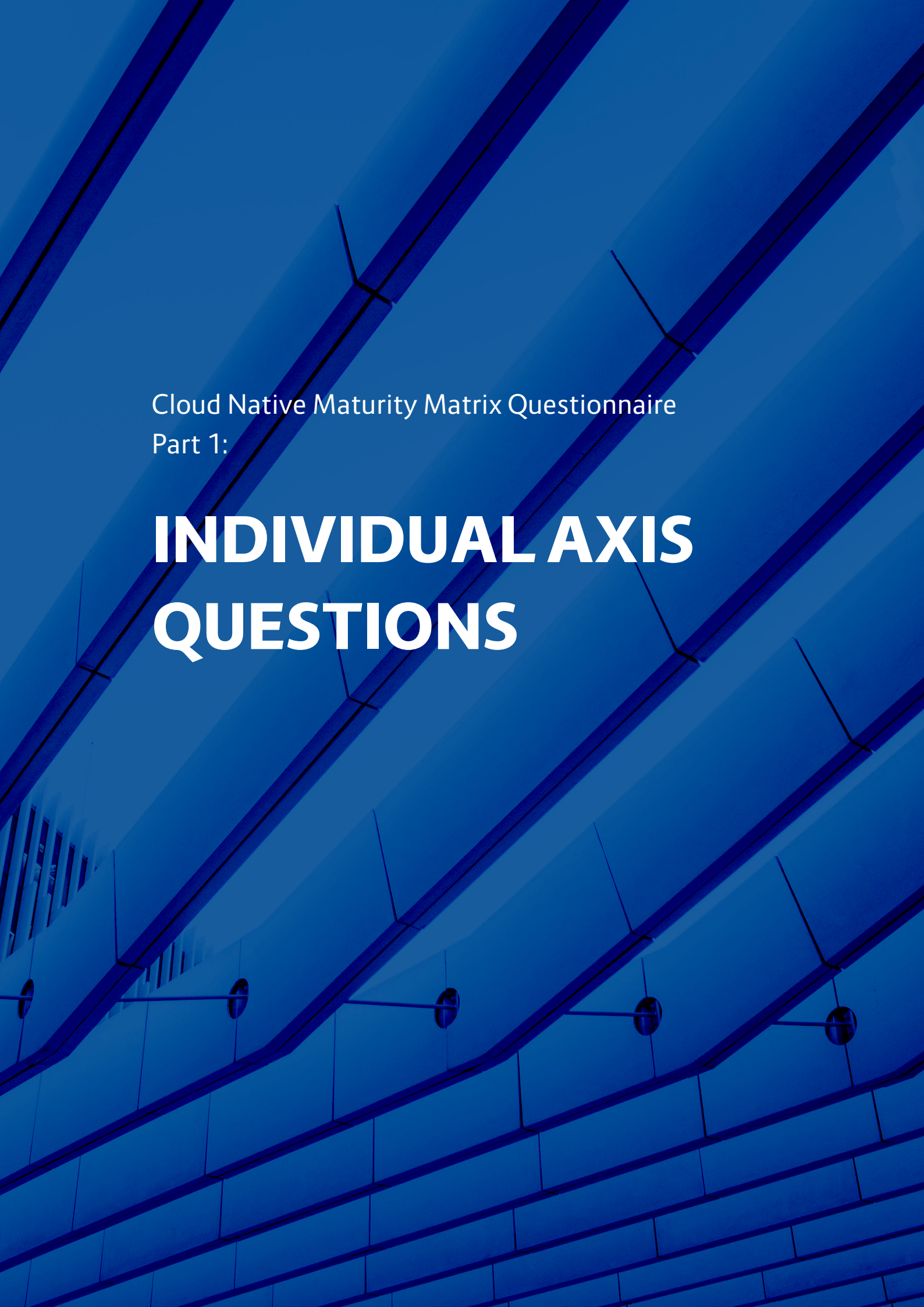
First, you will answer a series of questions designed to reveal your company's current practices, from Culture to Process to Infrastructure. We describe each of these areas and then ask you to choose the statement(s) that are reasonably true for your organisation. Mark your selections on the specific matrix section provided on each page. If an 'X into Y' result is indicated, fill in halfway between the two.

Stage	NO PROCESS	WATERFALL	AGILE	CLOUD NATIVE	NEXT
CULTURE	Individualist	Predictive	Iterative	Collaborative	Experimental

Next, copy your answers from the individual axes onto the full blank matrix. (A blank Maturity Matrix is provided on page 16). We use the answers to draw a point on each of the nine separate axes, and then literally connect the dots by drawing a line through each status point. Graphing status in this way gives instant valuable feedback, and provides a powerful visual of your company's current state.

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	All 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

We will then use this insight to match your results to typical scenarios that we have observed at other enterprises seeking to migrate their legacy systems and transform into a true Cloud Native entity.

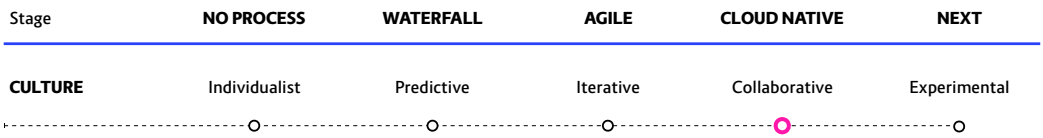


Cloud Native Maturity Matrix Questionnaire
Part 1:

INDIVIDUAL AXIS QUESTIONS

1. CULTURE

How individuals in your organisation interact, communicate, and work with each other.

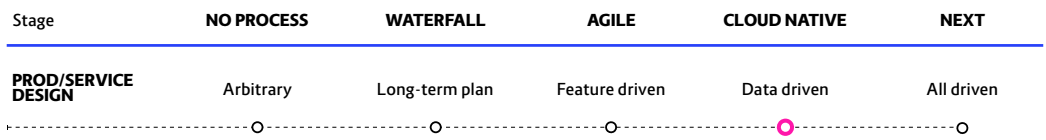


In our company ...

1. We decide what to do, and how to do it, on a project by project basis. (No Process/Individualist)
2. We build by deciding (for example) 'This feature in January and this in May and this in August' and then expect to see exactly those features on that schedule. (Waterfall/Predictive)
3. We have frequent meetings on a regularly scheduled basis. (Waterfall/Predictive into Agile/Iterative)
4. In our company, each team has clearly defined responsibilities. When a task is completed, it gets handed off to the next team in the production pipeline. (Agile/Iterative)
5. Our development teams focus on achieving small, defined objectives quickly. (Agile/Iterative into CN/Collaborative)
6. We have broad, high-level goals but are trusted to figure out the best way to implement them. (CN/Collaborative)
7. Failure is an option, so long as you learn from it. (CN/Collaborative into Generative)
8. Every team is empowered to choose, purchase, and implement the specific tech tools/services they need to independently develop and deploy applications/features/services. (Generative)

2. PRODUCT/SERVICE DESIGN

What you do, and how you go about doing it. How do decisions get made in your organisation about new products to build, or services and features to offer—or how to improve existing ones?

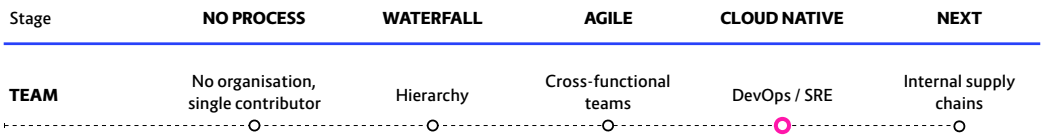


In our company ...

1. We release whenever the product is ready to ship. (No Process/Arbitrary)
2. Our CEO/board decides what is to be done and then passes it down to everyone else for execution. (No Process/Arbitrary into Waterfall/Long-term plan)
3. We have product roadmaps spanning months or even years into the future. (Waterfall/Long-term plan)
4. We release large sets of related features all at once as comprehensive updates. (Waterfall/Long-term plan)
5. Our feature releases are regular—for example, "Do X in two months, Y in four months and Z in six months"—with no deviation. (Agile/Feature driven)
6. If New Feature Z is scheduled to arrive six months from now but customers are asking for it sooner, we can possibly do that. (Agile/Feature driven into CN/Data driven)
7. When we deliver, we monitor to see if the release is a genuine improvement over the previous feature/service. (CN/Data driven)
8. Our systems and tech are automated to optimise our existing products/services. (Next/AI driven)

3. TEAM

How do responsibilities, communication, and collaboration work between teams in your organisation?

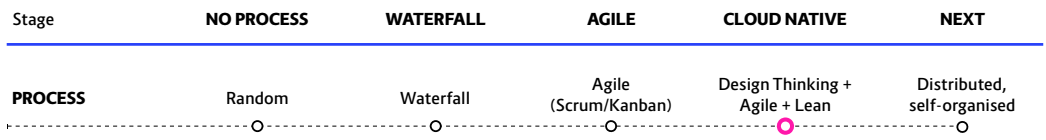


In our company ...

1. We have one tech team, and they handle everything. (No Process/Single Contributor)
2. We do all our planning up front, and then hand off to teams—often highly specialised—for execution. (Waterfall)
3. A team will work on one small, defined project and deliver it in two to four weeks. (Agile/Scrum)
4. When our piece of a project is finished, we hand it off to Operations for deployment. (Agile/Cross-functional teams)
5. Our teams are cross-functional—we have a designer, a UI expert, testing, someone with server experience, all in one group. (Agile/Cross-functional teams)
6. Our teams are meant to be self-sufficient and independent. We are tightly knit internally, but we don't talk to other teams very much. (Agile/Cross-functional)
7. Our developers build and then deploy independently, with no handover to a separate Ops department. (CN/DevOps SRE)
8. Our teams can communicate via APIs on a public cloud. (CN/DevOps SRE into Next/Internal Supply Chains)
9. We are not a small company, but we don't actually have tech teams. (Next/Internal Supply Chain)

4. PROCESS

How do you plan and then execute work?

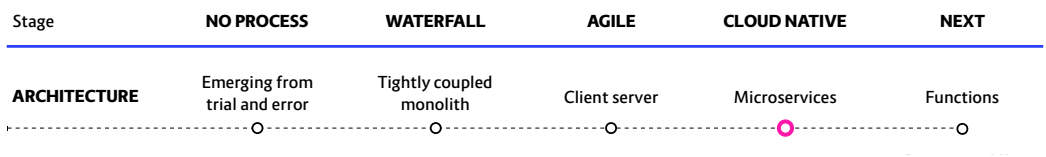


In our company ...

1. We change things on the fly, as necessary. (No Process/Random)
2. All decisions are made by managers. (Waterfall/Hierarchy)
3. When our piece of a project is finished, we hand it off to the next team. (Waterfall/Hierarchy)
4. It's difficult to give feedback to anyone outside my team. (Waterfall/Hierarchy to Agile/Scrum)
5. We run development as short-term, concentrated projects. (Agile/Scrum)
6. We focus on moving fast, so it's hard to do more than quick fixes when a bug or some other problem pops up. (Agile/Scrum)
7. Experimentation is encouraged, and we actually have time to do it. (CN)
8. We use different processes at different stages of product development as needed. (CN/Design Thinking into Next/Internal Supply Chain)
9. Our development process is fully automated so we have really small dev teams, if we have them at all. (Next/Self-organised)

5. ARCHITECTURE

What is the overall structure of your technology and systems?

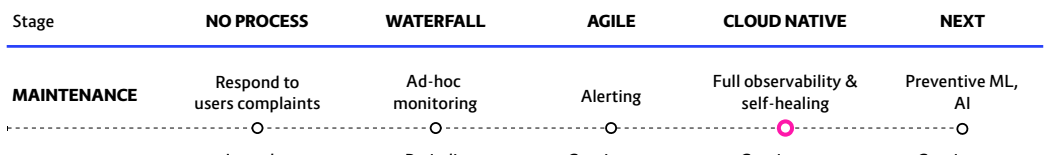


In our company ...

1. No one really knows the structure, we just build/buy what we need when we need it. (No Process/Trial and Error)
2. We fear the domino effect: if you change something, you have to be very very careful because it could break something else. (Waterfall/Monolith)
3. Very few people actually understand our entire system. (Waterfall/Monolith)
4. When we deliver, everything is delivered together, all ready on the same day and at a uniformly high level of quality. (Waterfall/Monolith)
5. Our application(s) is(are) divided into components, probably no more than five or six, communicating through networking. (Agile/Client server)
6. The size of an app in development is defined by the deployment schedule. (Agile)
7. Our applications share a single database. (Agile)
8. Our applications don't share databases. (Cloud Native/Microservices)
9. Our applications are broken into many small modular pieces, which communicate through APIs (Cloud Native/Microservices)
10. Our entire system is serverless and running on a fully managed cloud platform—our developers just write small functions. (Next/Functions architecture)

6. MAINTENANCE AND OPERATIONS

How does your organisation deploy software and run it in production?

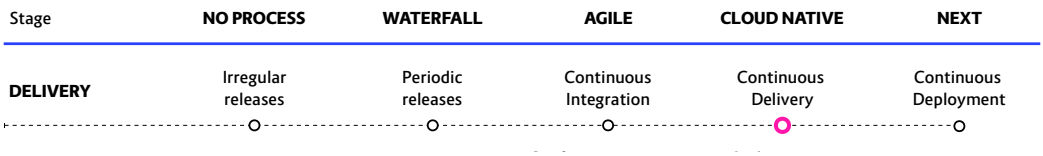


In our company ...

1. We only hear about problems, outages, or failures when users complain. (No process/Complaint driven)
2. We have some simple automation, like scripts, for alerting large-scale issue and outages in the field. We find out about smaller problems from user reports. (Waterfall/Ad hoc).
3. When we find a problem, we fix it manually—someone from Operations logs into a production server and follows a preset procedure. (Waterfall/Ad hoc)
4. Our systems have full and continuous monitoring, and our Ops team spends lots of time checking on alerts. (Agile/Alerting)
5. When an exception gets thrown in our containerised application, by the time we log in to see what happened there is no trace of the problem. (Agile/Alerting)
6. Ops has security access to production servers because sometimes they need to view information on a certain machine. (Agile/Alerting)
7. If a server crashes, our system has the ability to detect this and restart it. (CN/Full observability and Self-healing)
8. We have a dashboard/UI where anyone in the organisation can see a full overview of system status at any time. (CN/Full observability)
9. Our system is able to monitor itself automatically, detect all issues, and fix itself when things go wrong. (Next/Machine Learning/AI)

7. DELIVERY

How does software progress from your development teams to running live in production?

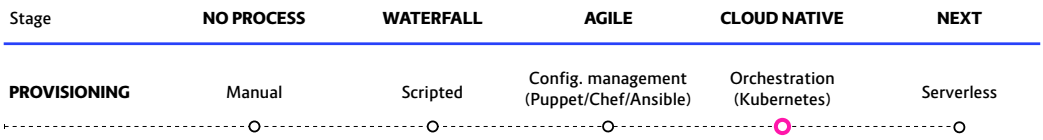


In our company ...

1. We deliver software whenever a new release/version is needed/ready. (No Process/Irregular releases)
2. We do major version releases, very carefully planned and scheduled, every six to 12 months. (Waterfall/Periodic releases)
3. We don't like to make changes to our production code, even emergency ones, because there are so many dependencies. Change is risky. (Waterfall/Periodic)
4. A lot of planning goes into our next release before any actual development begins. (Waterfall/Periodic releases)
5. Our delivery process includes some test automation and automated build, but outside of final integration. (Agile/Continuous Integration)
6. We work on feature branches, which we integrate into the main line only periodically. (Agile/Continuous Integration)
7. Code quality is consistent and high enough that we can release at any given moment. (CN/Continuous Delivery)
8. Tech teams typically integrate and push new code to live production servers every day. (CN/Continuous Delivery)
9. Emergency patches are just code changes like any other, no big deal. (CN/Continuous Delivery into Next/Continuous Deployment)
10. When a developer checks in code, it goes directly into production. (Next/Continuous Deployment)

8. PROVISIONING

How does your organisation create and then control new infrastructure? How quickly can you deploy?

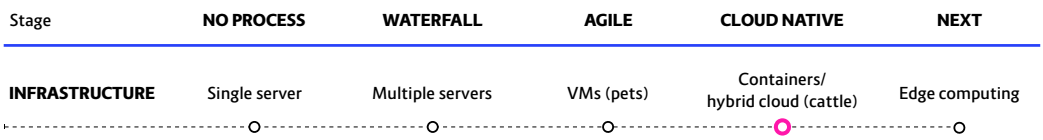


In our company ...

1. We set everything up manually and then run it ourselves too. (No process/ Manual)
2. Operations team is in charge of provisioning, period. (Waterfall/Scripted)
3. Developers write applications, and specify what they will need to run successfully in production (OS, libraries, dependent tools). (Waterfall/ Scripted)
4. The Ops team manually configures the production machines to meet the machine dependencies the Dev team specified. (Waterfall/Scripted)
5. We might have a few scripts to help with config. (Waterfall/Scripted into Agile)
6. It takes hours or days to provision a machine, which is fully automated (maybe even auto provisioned) by Ops. (Agile/Config Management)
7. Provisioning is a mix of automation and manual work. (Agile/Config Management)
8. Developers write, and test, containerised applications. (CN/Orchestration)
9. We use an orchestrator, like Kubernetes, to automatically spin up and run containerised applications as needed. (CN/Orchestration)
10. Data centers are so last decade! We do our data processing on the edge! (Next/Serverlesscomputing)

9. INFRASTRUCTURE

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



In our company ...

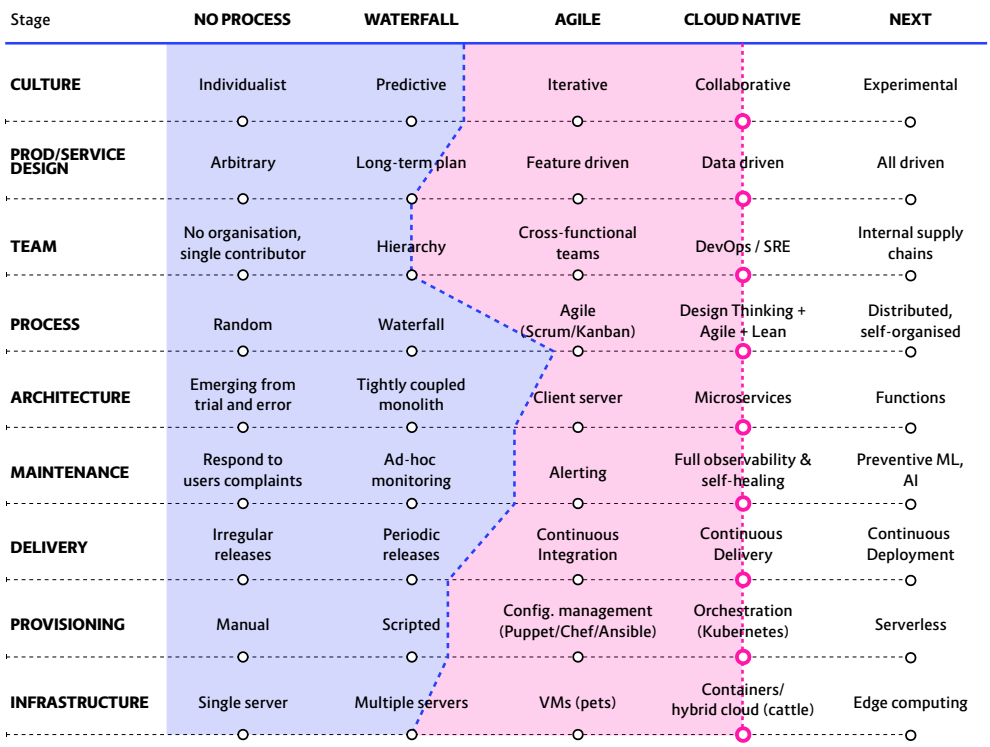
1. Everything pretty much runs on one single server ... which may or may not be under somebody's desk. (No Process/Single Server)
2. We have multiple physical servers in our own private data center (either on premises or co-located). (Waterfall/Multiple servers)
3. If one of our servers goes down, we have to manually provision its replacement. (Waterfall/Multiple servers)
4. We don't use physical servers—we have VMs. (Agile/VMs)
5. Provisioning infrastructure is a mix of automation and manual work, so a new VM can take a couple of days to set up. (Agile/VMs)
6. We also have some instances in the cloud, which we manage manually. (Agile/VMs into CN/Containers & Hybrid cloud)
7. Our infrastructure is fully automated and we can provision in seconds. (CN/Containers & Hybrid Cloud)
8. We use a managed solution from our public cloud provider and trust them to handle the infrastructure's finer details. (CN/Containers & Hybrid Cloud)
9. Our engineers and developers can fully self-service infrastructure on their own. (CN/Containers & Hybrid Cloud)

DRAW YOUR LINE

Use your answers from each of the individual axis questions on the previous pages to draw a point on each of the nine separate areas on the full Maturity Matrix chart. If you chose more than one answer for a particular axis, mark each one in the appropriate spot. Now literally connect the dots by drawing a line through the status point marked on each axis.

If you have more than one point on a particular axis, please draw the line through the one you feel best fits your situation. (When in doubt, choose the one lying farthest to the left).

Remember our sample matrix showing real-world results from an enterprise's Cloud Native assessment?

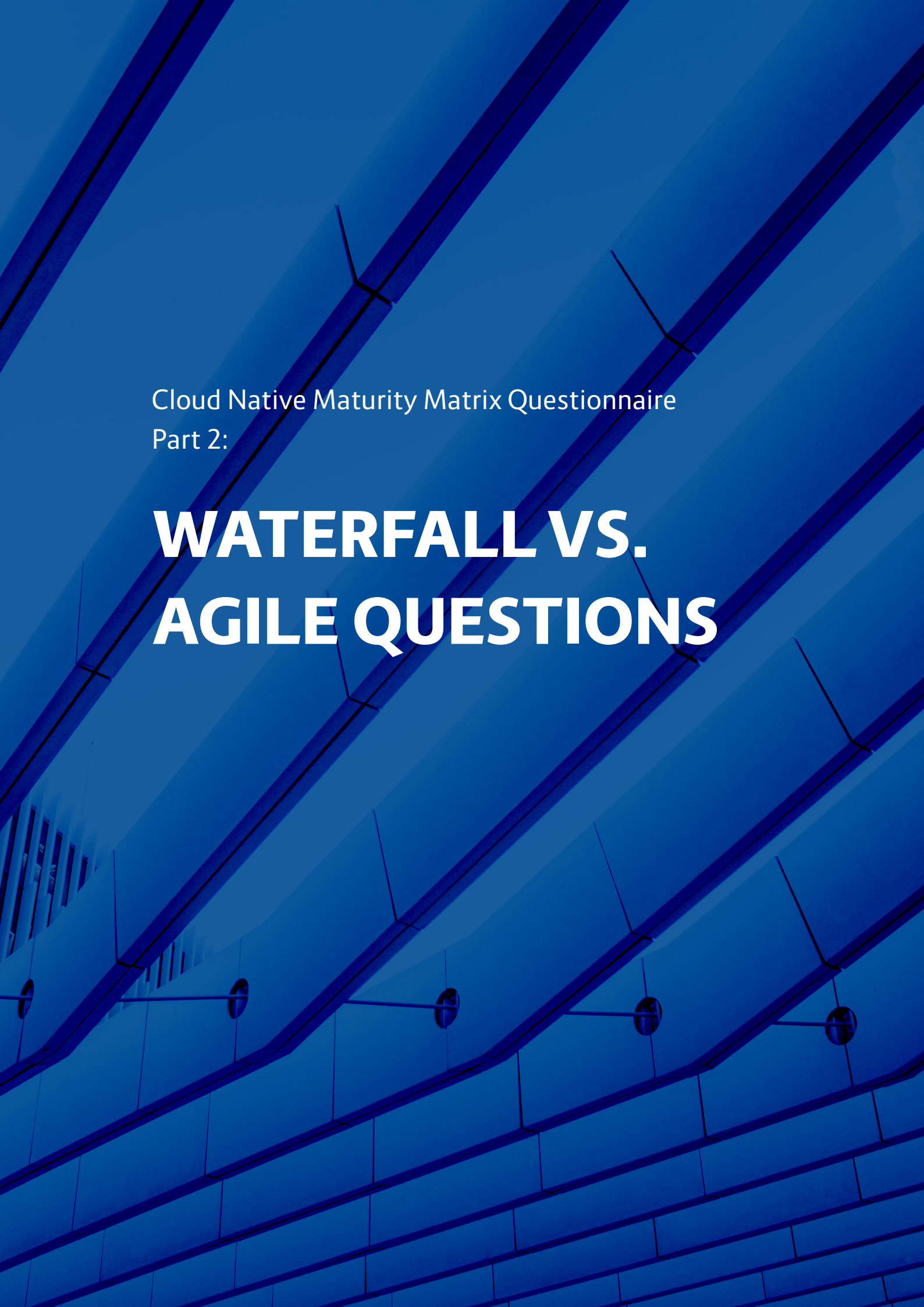


It shows a company that is functioning largely in a traditional Waterfall hierarchy, especially in the Team, Design, Provisioning and Infrastructure areas.
Culture, however has progressed somewhat and is starting to move toward Agile; less predictive, more iterative.

Perhaps this was in response to advances in their Process and Architecture, which have nearly reached Agile state: we would gauge from this that they are using Scrum sprints to deliver more frequently.

CLOUD NATIVE MATURITY MATRIX

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	All 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



Cloud Native Maturity Matrix Questionnaire
Part 2:

WATERFALL VS. AGILE QUESTIONS

WATERFALL VS. AGILE

There is a great deal of talk about the crucial role culture plays in Cloud Native. But what does that really mean, and why is it so important?

Culture is the sum of the daily actions that you take. Your routines. If you talk to people you have collaborative culture. Needing to seek permission before trying something new means you have hierarchical culture. If you change the actions, you change the culture.

You can't have Cloud Native culture but not have Microservices—if it takes you six months to deliver, you can't be distributed. Can you do DevOps in a Waterfall organisation? Sure. On the Dev side you can apply Cloud Native practices to optimise and accelerate your development process. Meanwhile on the Ops side you can automate deployment to make your provisioning and infrastructure faster and more efficient than ever before. This is all great ... except for the fact that your beautiful containerised microservices and their state-of-the-art platform won't actually get in front of users until the next release cycle is completed, many months from now. All that speed and efficiency? Simply wasted. Yes, you are doing CN right—from inside Waterfall.

Cloud Native is not automatically the 'right' solution, and having a Waterfall culture is not necessarily wrong or 'bad.' There are times when Waterfall is absolutely the best process for the situation at hand. Agile, too, has circumstances where it is the optimal choice. The problem comes when the wrong solution gets applied—or a combination of solutions that conflict, undermine, and ultimately gridlock each other.

This is why culture matters. And why understanding your own organisational culture is critical for functioning well in the world—i.e., succeeding as a business.

The first step in understanding these forces that shape your organisation is to examine the actions that define your day-to-day operation: Know Thyself. Since the vast majority of software is built using Waterfall and/or Agile practices, we have developed two additional batteries of questions investigating how your organisation functions in different areas. The answers indicate the most likely culture alignment.

WATERFALL

Please answer Agree, Disagree, or Neutral/Not Applicable for each statement.

In our company ...		Agree	Neutral	Disagree
1.	We release large sets of related features all at once as comprehensive updates.	○	○	○
2.	We know the next step in our process and when it is going to happen.	○	○	○
3.	I know who my boss is, and my boss's boss.	○	○	○
4.	Documentation is a priority because that is how other teams (and managers) know what we are doing.	○	○	○
5.	Specialist teams handle specific areas: design, architecture, security, testing, etc.	○	○	○
6.	We often talk in JIRA issue numbers.	○	○	○
7.	Our system is so complex that very few people understand the entire thing.	○	○	○
8.	Once we release a software version all changes have to wait for the next version roll out months, or even a year, from now.	○	○	○
9.	We don't like to make changes to our production code, even emergency ones, because there are so many dependencies. Change is risky.	○	○	○
10.	We have lots of planning sessions with other teams.	○	○	○
11.	Business analysts play a big role in our software-delivery process.	○	○	○
12.	We do 'big bang' releases that roll lots of changes into one new version.	○	○	○
13.	We increase quality by building abundant time for testing into the build process.	○	○	○
14.	You have to open a ticket to provision a machine, engineers can't self-service.	○	○	○
15.	Developers write applications, and specify what they will need to run successfully in production (OS, libraries, dependent tools). Then the Ops team configures the production machines to meet the machine dependencies the Dev team specified.	○	○	○
16.	We have multiple physical servers in our own private data center (either on premises or co-located).	○	○	○

AGILE

Please answer Agree, Disagree, or Neutral/Not Applicable for each statement.

In our company ...		Agree	Neutral	Disagree
1.	Our releases are usually small-scale iterative changes to existing features/services.	○	○	○
2.	Our feature releases are regular—for example, 'Release Feature X in two months, Feature Y in four months and Feature Z in six months'—with little or no deviation.	○	○	○
3.	Our team communicates well internally, but we don't talk to *other* teams very much.	○	○	○
4.	We run development as short-term, concentrated project 'sprints'.	○	○	○
5.	Inside of our teams, we have great mutual support and cameraderie.	○	○	○
6.	We focus on moving fast.	○	○	○
7.	Hard to do more than quick fixes when a bug or some other problem pops up.	○	○	○
8.	Our applications are divided into components that communicate through networking.	○	○	○
9.	Our individual teams each have very vertical and targeted responsibilities—we handle exactly one slice of the pie (a single component).	○	○	○
10.	New functionality requests can usually be accommodated within a few weeks.	○	○	○
11.	Developers test and merge their changes every few days, but not directly onto the production system.	○	○	○
12.	We don't use physical servers—we have VMs.	○	○	○
13.	Any task taking longer than a week to provision to VM breaks the production cycle, and so is a nonstarter.	○	○	○
14.	Sometimes you can be a hero by responding fast and solving a crisis.	○	○	○
15.	Our teams are cross-functional. For example we can have a designer, a UI expert, testing, someone with server experience, all in one group.	○	○	○
16.	There is not much point in interacting with customers because too much time passes before we could give them whatever they're asking for.	○	○	○

WATERFALL VS. AGILE ANSWER KEY

Scoring: 1 point for Agree, 0 for Neutral/Not Applicable, -1 point for Disagree

Waterfall Results:

5-10 points indicates a likelihood of strong Waterfall culture

1-4 points indicates areas of Waterfall culture are likely present (Mixed culture, such as Waterfall moving to Agile—you may have Scrum, for example)

0 and below indicates high likelihood Waterfall culture is not present

Agile Results:

5-10 points indicates a strong likelihood of Agile culture

1-4 points indicates areas of Agile culture are present; Mixed culture likely, such as Agile with CN elements like MS or CI/CD

0 and below indicates high likelihood Agile culture is not present

CONGRATULATIONS

You have completed a simplified and abridged version of the same diagnostic tool Container Solutions engineers use when we are called in for an assessment. (Ours has many, many more questions, takes two days to fully administer, and includes focus group and individual interviews for fact finding and maximum depth).

We hope you found the process helpful and maybe even enjoyable. But a word of caution: this is meant to provide a very general top-level overview. Cloud migrations are very complex (and expensive) undertakings. Please be sure to consult with experts who have previously—and successfully—navigated the Cloud Native transformation path

A good next step: book a Container Solutions Cloud Native Readiness Assessment. Following our multi-day site visit we provide a full report of findings and a thorough analysis.

Our co-founder and CEO Jamie Dobson promises, 'You will never regret bringing us in for a discovery. Investing in two days of organisational self discovery gives hugely valuable insights into your company's current situation. And, should you decide to undertake a Cloud Native transformation, it will save you a lot of time—not to mention a lot of money.'

Cloud Native Maturity Matrix Questionnaire
Part 3:

A GLOSSARY OF CLOUD NATIVE TERMS

A GLOSSARY OF CLOUD NATIVE TERMS

AGILE: Agile methodology is a widely used approach to project management in software development. It is based on using incremental, iterative work sequences that are commonly known as sprints. Scrum is a subtype of agile methodology, essentially a specific framework for agile software development.

CLOUD: Cloud computing, or 'the cloud' is a general term for delivering hosted services over the internet (the name being inspired by the cloud icon often used to represent the internet on diagrams/flowcharts). Cloud services are different from traditional infrastructure and platform hosting because they are elastic (users can utilise as much, or as little, service as they need at any given moment); sold on demand (by the minute or the hour, rather than set contract allotment); and fully managed by the provider (the consumer needs nothing but computer and Internet access). Significant innovations in virtualisation and distributed computing have accelerated interest in even smaller enterprises moving to cloud computing. Public versus Private: public clouds like Google and Amazon Web Services sell services to anyone, and all users share the same resource pool. A private cloud is a proprietary network or data center, usually company-owned, with access limited to specific entities.

CONFIGURATION MANAGEMENT: Specifically, the automation of server configuration and management, using tools such as Ansible, Puppet, Chef, or Terraform.

CONTAINERS: Containers are lightweight, standalone executable software packages that include everything required to run an application: code, runtime, system tools, libraries, and settings. They are a sort of 'standard unit' of software that packages up the code with all its dependencies so it can run anywhere, in any computing environment. You can think of them as scale-able and isolated VMs in which you run your applications. You can link containers together, set security policies, limit resource utilisation, and more.

CONTINUOUS INTEGRATION: Continuous Integration is a coding philosophy and set of practices that drive development teams to implement small changes and check in code to version control repositories frequently. The technical goal of CI is to establish a consistent and automated way to build, package, and test applications.

CONTINUOUS DEVELOPMENT: Continuous Delivery starts where Continuous Integration ends. CD automates the delivery of applications to selected infrastructure environments. Most teams work with multiple environments outside the production pipeline, such as development and testing environments, and CD ensures there is an automated way to push code changes to them. CD automation then performs any necessary service calls to web servers, databases, and executes procedures when applications are deployed.

CI/CD: When integrated, CI/CD together make it possible to implement continuous deployment where application changes run through the CI/CD pipeline; passing builds get deployed directly to production environments. Teams practicing continuous delivery elect to deploy to production on daily or even hourly schedules.

CROSS-FUNCTIONAL TEAMS: A cross-functional team is where members have different skill sets and competencies, but are working collaboratively toward the same goal. Team members have all competencies necessary for accomplishing the work within the team—so there are no dependencies on others outside the team. In software development this typically means front-end and back-end developers, database and UX specialists, QA engineers, and any other role necessary for producing the product/service.

CULTURE: How individuals within an organisation communicate and work with each other. Culture is the sum of the daily actions that you take. Your routines. If you talk to people you have collaborative culture. Needing to seek permission before trying something new, you have hierarchical culture. If you change the actions, you change the culture.

DATA-DRIVEN DESIGN: Data-driven design describes the practice of developing or improving a product based on things you can measure. Metrics like site analytics, carrying out A/B testing, or surveying users for feedback are all used to make design decisions.

DESIGN THINKING: Design thinking is a human-centred approach to business processes. It focuses on customer problems and challenges as the foundation to producing products/services that satisfy their wants and needs.

DEVOPS: DevOps is both a culture and set of processes aimed at reducing the division between software development and its actual operation. With DevOps, the traditionally siloed Development and Operations teams work together as one cohesive team (or, sometimes, two teams in tight collaboration). The approach facilitates fast and seamless software development while optimising both productivity and reliability. Regardless of organisational structure, companies adopting the DevOps model create teams that embrace the entire development and infrastructure lifecycle in their scope of responsibility. See: SRE.

GREENFIELD PROJECT: Greenfield deployment refers to building a complete software development system where previously there was none. In Cloud Native it means not just cloud-based infrastructure but also incorporates architecture, design, process, and culture—basically, starting completely from scratch in every possible area. The term comes from the construction industry, where greenfield development refers to any project on pristine, previously undeveloped land. Greenfield development is often viewed as advantageous because it is free from constraints that can be imposed by a system's existing networks/infrastructure or other legacy elements.

MICROSERVICES: Microservices (microservice architecture) is 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.
Microservices enable development teams to take a more decentralized (non-hierarchical) approach to building software. Microservices enable each service to be isolated, rebuilt, redeployed, and managed independently.

MONOLITH: 'Monolith' is used to describe a single-tiered software application in which different components combine into a single program, launched from a single platform. A monolithic application is self-contained, and independent from other computing applications, its design based on a 'batteries included' philosophy that makes the application responsible not just for one particular task, but can perform every step needed to complete any function. Monolithic apps are huge and complex, and therefore very slow to deliver changes/supdates. A typical monolith development cycle is six months to one year.

ORCHESTRATION: Orchestration in general refers to the automated configuration, coordination, and management of computer systems and software. In cloud computing, it refers more specifically to Kubernetes, an open-source system for automating the deployment, scaling, and management of containerized applications. In Cloud Native, orchestration is all about managing the lifecycles of containers, especially in large, dynamic environments. (Dev)Ops teams use container orchestration to control and automate tasks such as the availability, provisioning and deployment of containers, load balancing of containers across infrastructure, and scaling up/down by adding/removing containers as needed. See also: Containers.

SRE: SRE, which stands for SIte Reliability Engineering, evolved independently of the DevOps movement but fits perfectly with it. SRE embodies the DevOps philosophy and then takes it one step further to define an architecture for implementing them. SRE offers a much more prescriptive way to measure and achieve reliability across the full spectrum of DevOps responsibilities. DevOps is a philosophy; SRE is a set of practices. SRE is typically the desired model in very large frameworks, like Google.

WATERFALL: The waterfall model of software development is based on a logical progression of steps that form the software development life cycle (SDLC). One follows after the other in strict order, much as a waterfall cascades down from top to bottom. While more agile methodologies have arisen, causing the waterfall model to decline in popularity, waterfall's sequential process still contains advantages and it remains a common design process in the industry.