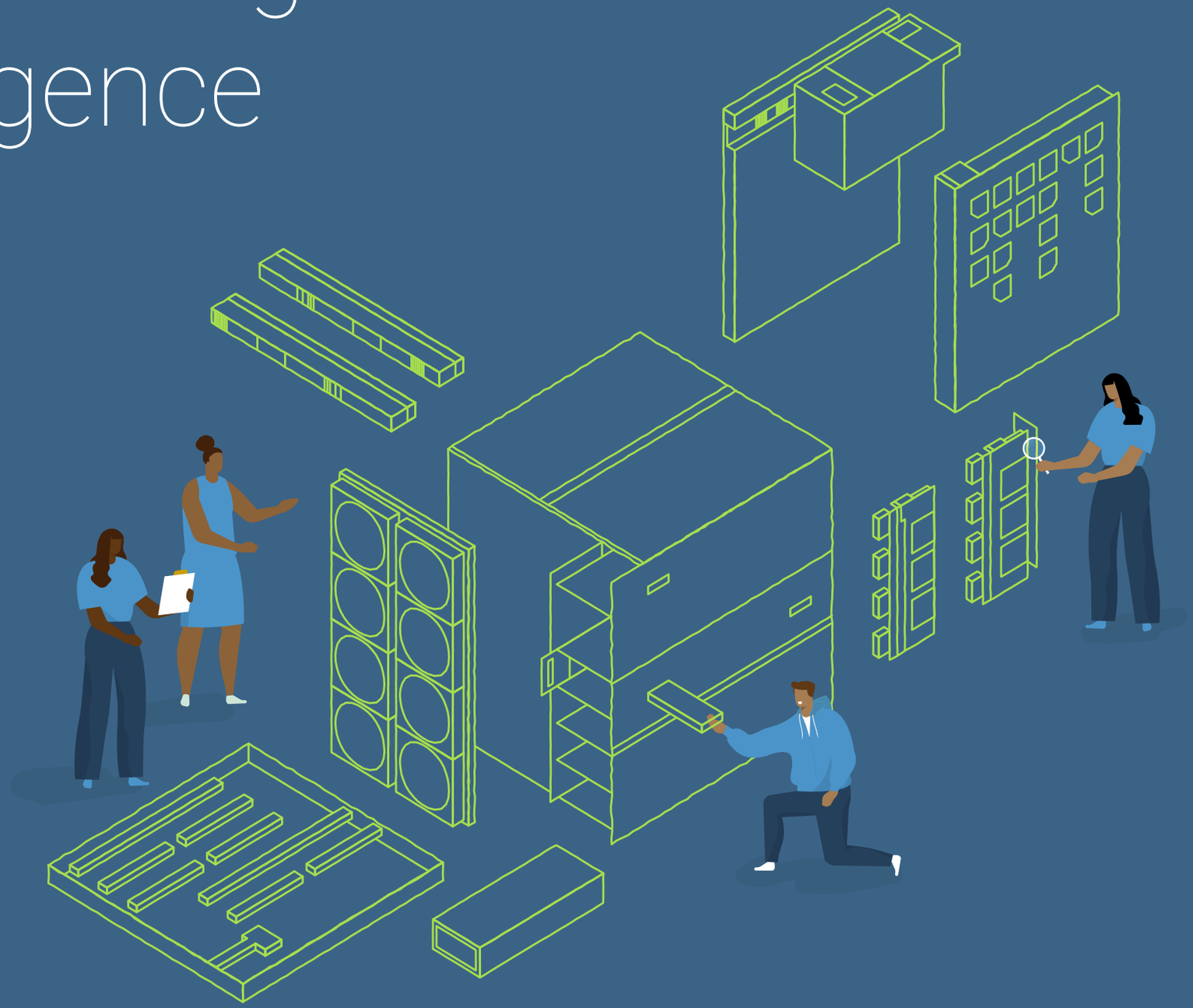
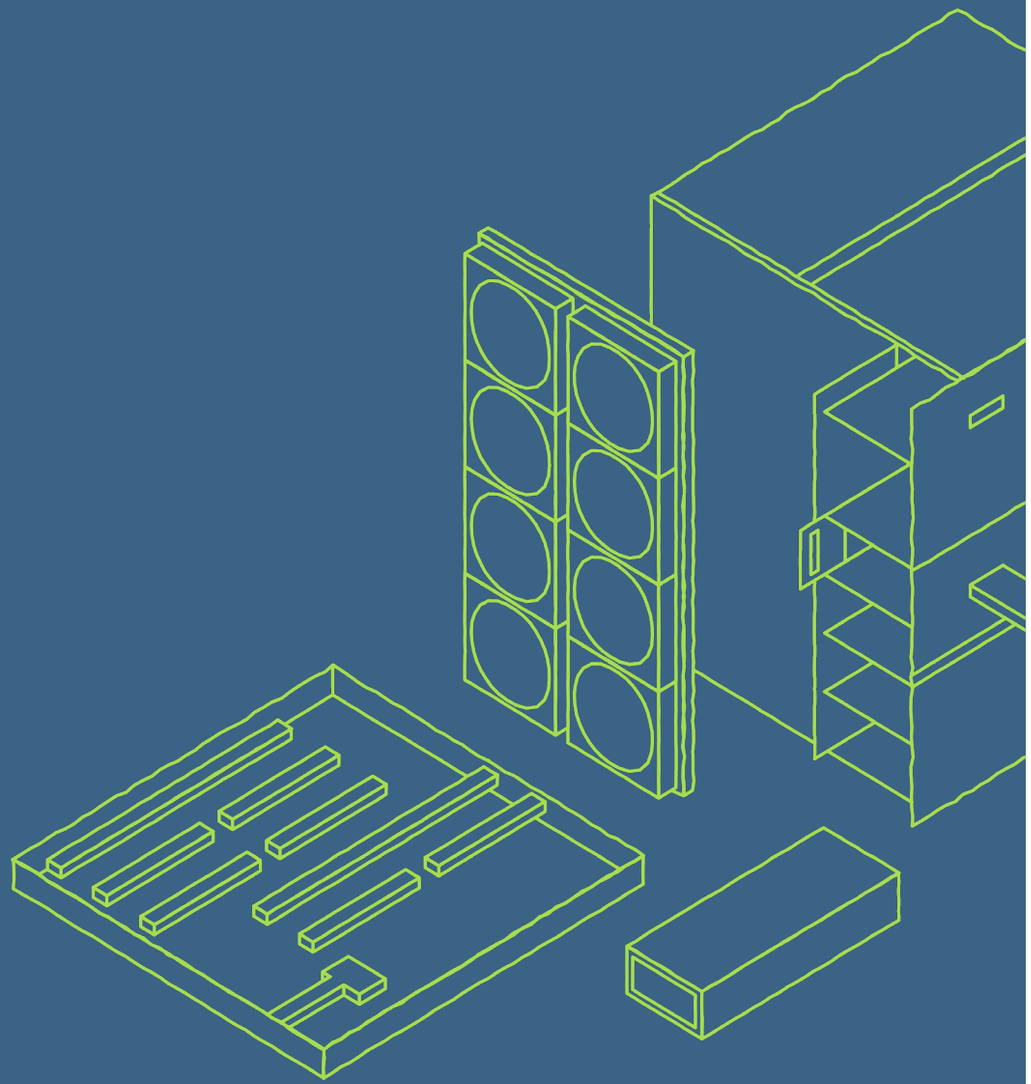
A Guide to Surviving Tech Due Diligence

BY JOHANN ROMEFORT



circleci

Introduction



If you're building a technology company, at some point, whether your company is seeking funding or a potential M&A, you'll be subjected to the exercise of Due Diligence. Due Diligence is a process where the interested party will want to assess the risk associated with the transaction. During Due Diligence, several experts will be deployed to audit nearly every facet of the company, including all technology-related aspects.

Throughout the years, I've been involved in auditing many companies, conducting tech Due Diligence for VCs, Private Equity funds, and corporations. On many occasions I've witnessed how ill-prepared some CTOs have been; anxious and not at the top of their game. If you try to find information online about the Tech DD process, chances are you'll come across articles that only touch on a few points, leaving most people wondering what they really need to prepare for.

This Tech DD guide will help you understand not only the process itself, but more importantly, what topics will be addressed, which documents you'll have to produce, and how to prepare mentally for such an exercise.

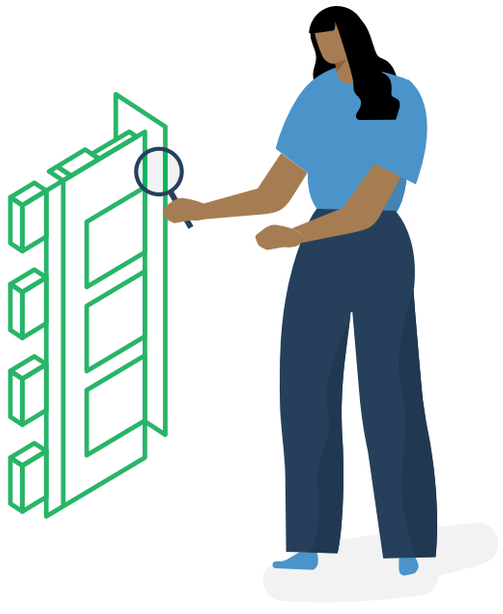
What you'll find throughout this guide is not only how to succeed at this exercise, but also advice on what it takes to build a balanced technology platform and team.

About the author

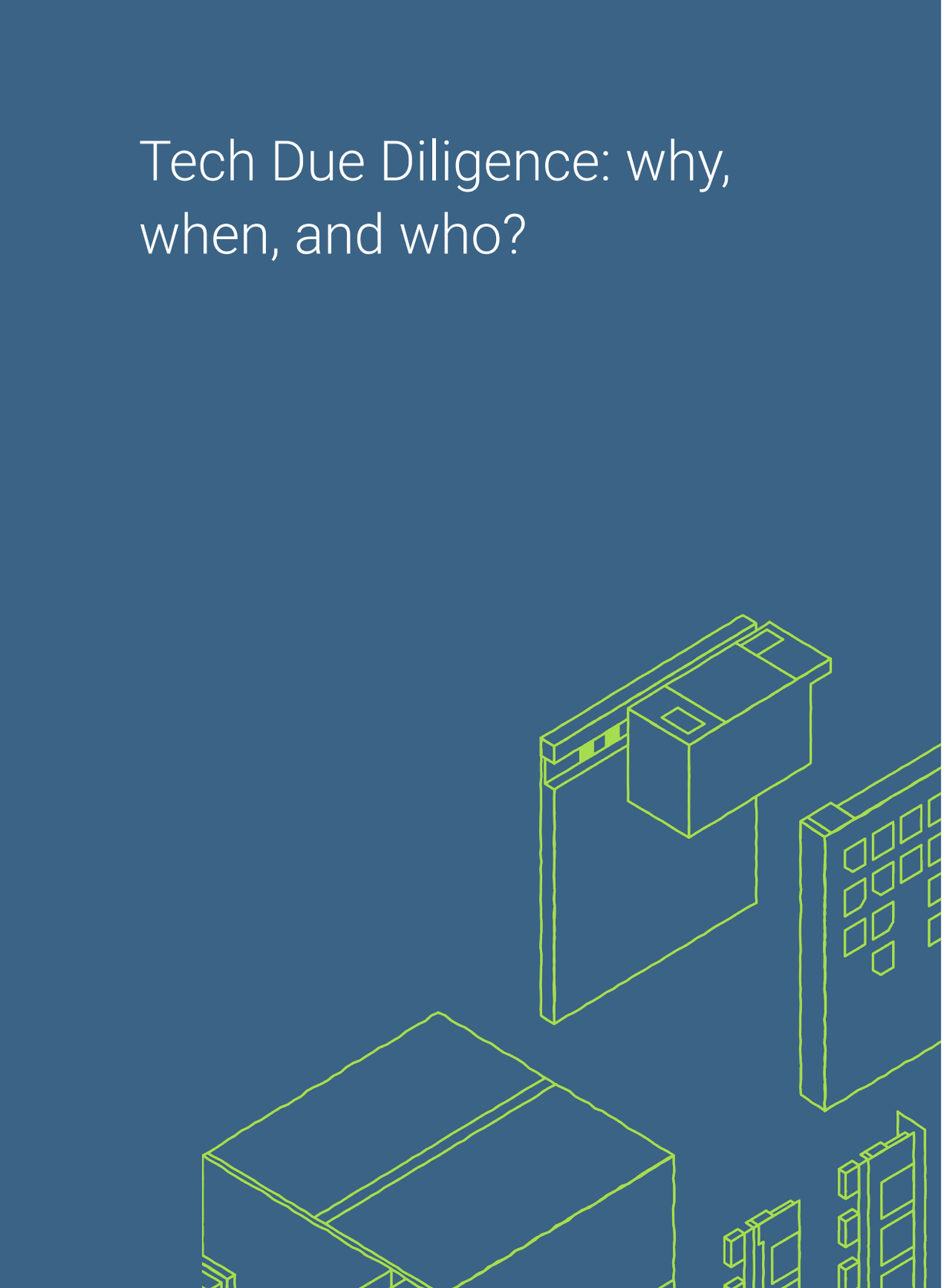
My first experience with Tech Due Diligences was when I was the CTO at Seesmic, a SaaS company I co-founded. Back then I had no idea what to expect when my CEO said that someone was coming to ask me questions about what I built and about the tech team. It turned out that this “someone” was the former CTO and co-founder of LinkedIn. While very excited by the idea of talking to someone of that caliber, I was also quite anxious because the information available on the topic of Tech DD was very sparse.

More than a decade later, I ended up on the other side of the table. First, as a Managing Director at Techstars, investing in 10 companies yearly, then as a CTO coach, helping them to prepare for such exercises. More recently, I’ve worked as an independent consultant for VCs and Private Equity funds performing technical audits in the context of investments and M&A.

I’m now working on a training program aimed at CTOs, to help them prepare successfully for Tech DD and set them up for success.



Tech Due Diligence: why, when, and who?



It’s all about risk assessment

Let’s assume you’re buying a house. You’d do a bit more than have a quick look at the facade and interiors. Most likely you’ll inspect every corner of the house and you might even commission someone like an architect who can spot hidden flaws better than you can.

When it comes to technology transactions, the party who’s either investing in, buying, or even sometimes using (yes, your customer) your product, needs to understand what they’re getting into. How much risk are they taking? If they invest to scale the company, will you be able to scale, or will you spend part of this investment trying to rebuild fragile foundations that could crumble at any time?

The larger the transaction is, the more in-depth the assessment will be, as the stakes are higher. Generally, as soon as you hit your Series A, you’ll be faced with the process of Due Diligence.

Even though a Tech DD is not so common before the Series A stage of funding, there are investors out there who are more diligent at earlier stages or particularly concerned about the tech aspects. The complexity and thoroughness of the assessment will increase as you move up the investment series and possibly conclude with an acquisition.

Technical Due Diligence is generally commissioned by:

- Venture Capital firms
- Family offices
- Private Equity Funds
- Large corporations

If you're dealing with very large customers or partners, there is a chance that they also submit you to a Tech DD, particularly (but not only) in the fintech and health tech industries, where corporations want to be on the safe side when partnering with third-party providers.

I've been in your shoes before. When my company raised a major round of funding and I was informed that the former LinkedIn CTO would be coming to audit everything I built, I had no idea what to expect.

Looking at past, present, and future

Whenever an audit is performed, the auditor will start by trying to understand how the product was built from day one. They're trying to figure out how the product has evolved, what kind of compromises were made, and how fast the team is executing.

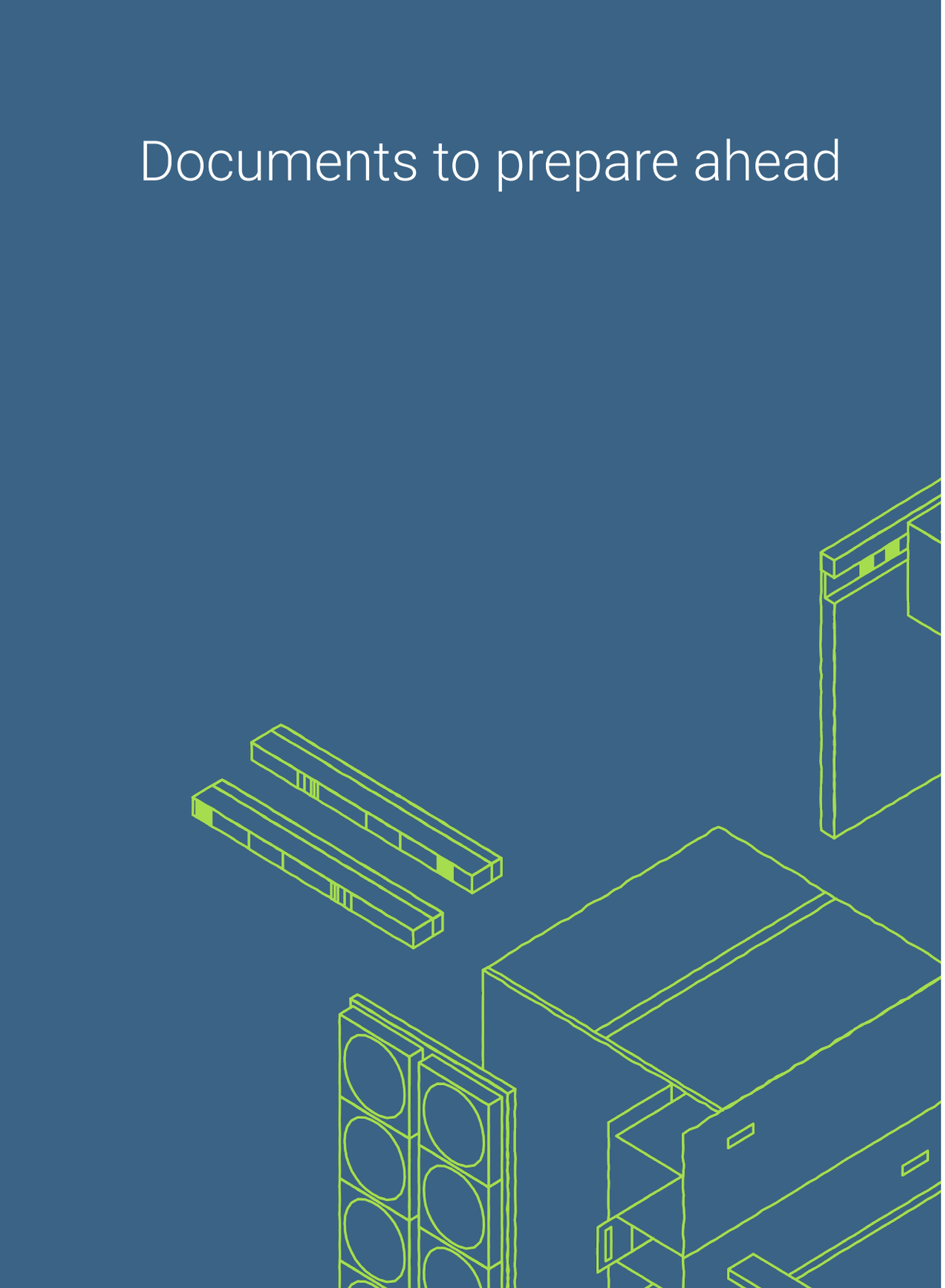
The auditor will pay particular attention to the different stages of major refactorings that happened during the life of the product. They'll want to know about:

- Initial architecture choice
- Was the technology entirely rebuilt at any point(s)?
- What sort of compromises were made?
- How does the team think about scalability?
- How does the team think about reliability vs. velocity of development?

The past generally informs the present and the future. Auditors like to understand what you've learned from past mistakes and how those mistakes have shaped the present. Be transparent, reflective, and do your best to communicate the challenges you've faced in getting where you are.

Auditors are generally former CTOs themselves. They understand the struggles of building a company and communicating your challenges is a sign of humility and transparency which is often appreciated and increases the empathy level of the auditor. Not only is transparency appreciated, but it also denotes someone's ability to solve problems faster by seeking help while being able to articulate, ego aside, the challenge one is facing.

Documents to prepare ahead



There are several documents that should be ready before the Tech DD starts. Don't wait for the process to begin before producing these documents as you'll most likely not have enough time to put sufficient thought into them. This could be detrimental to the outcome of the audit, as you'll be prone to make mistakes and forget important details while rushing through it

Org chart and resumes

An organization chart of the whole company with positions of each member, as well as the resume of each member of the tech team. This is used to assess the level of seniority within the team and how long each team member has been with the company.

OKRs (if applicable)

Objectives and Key Results or OKRs are a way to align a whole organization on the company's vision and mission. They are actionable by nature and generally implemented at the company, team, and individual levels. Looking at historical OKRs can show:

- How the company has evolved
- How often it changes gears
- Ambition
- Execution speed

Product roadmap and technology roadmap

The auditor will want to see how well the product and technology roadmaps align with each other. This is to detect potential discrepancies between product and technology, which could be a sign of misalignment between them — something that happens frequently.

Provide a view on the next six months, even if these roadmaps might change.

Development process documentation

These documents (which can be a mix of screenshots/exports from the tools you're using and simple Google Docs) describe how the software is built from request to production. It should show the following:

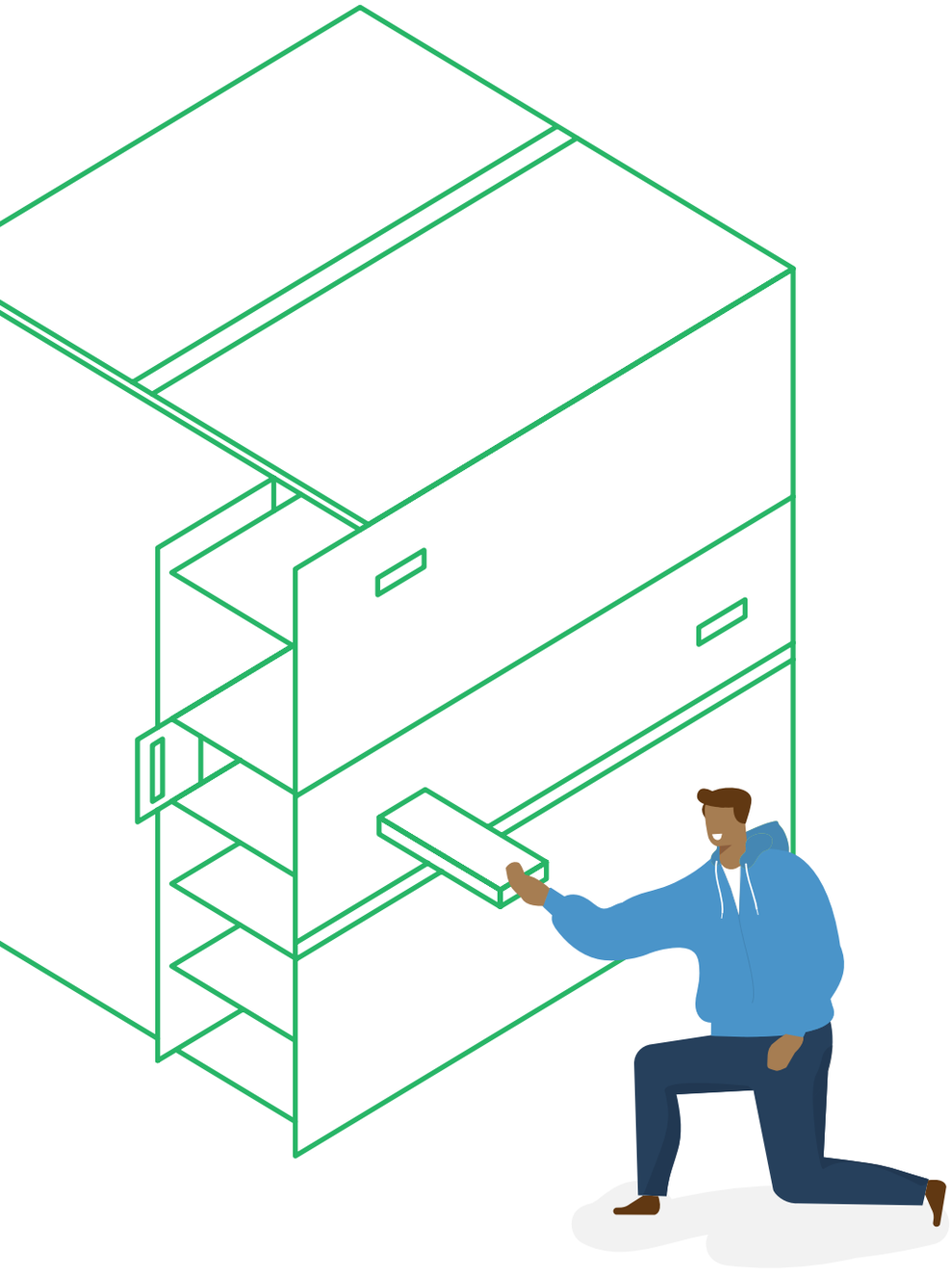
- Ticket creation for features and bugs
- Creation of branches in the source repo
- Pull requests
- Potential PR reviews
- CI/CD pipelines (builds, tests, vulnerability checks, code quality)
- UAT (User Acceptance Testing)
- The DevOps process to put code into production (ex: rebuild Docker images and update on the server).

Architecture diagrams

Architecture diagrams are the most important documents for the auditor to review when it comes to understanding the flow and complexity of your technical infrastructure. They're also a great way to align the whole tech team around a set of standardized documents. They will be particularly useful when you onboard new developers, so each person gets a comprehensive overview of every layer of the architecture and can get started as fast as possible in their new role.

Designing a great architecture document revolves around some core principles:

- It should describe the flow of information within the system.
- It should provide clear groupings of logical components and boundaries between them.
- It should include additional information in the form of annotations to provide a greater level of detail and understanding.



Ideally, you should create the following documents:

Application and Integration Architecture Diagram

This diagram should provide a high-level description of the main components in the system and how they interact with each other. Components should be grouped by layers, such as presentation (your UI), logical (the brain that orchestrates the flow of events and glues everything together), data processing (ML Services), data storage (your databases).

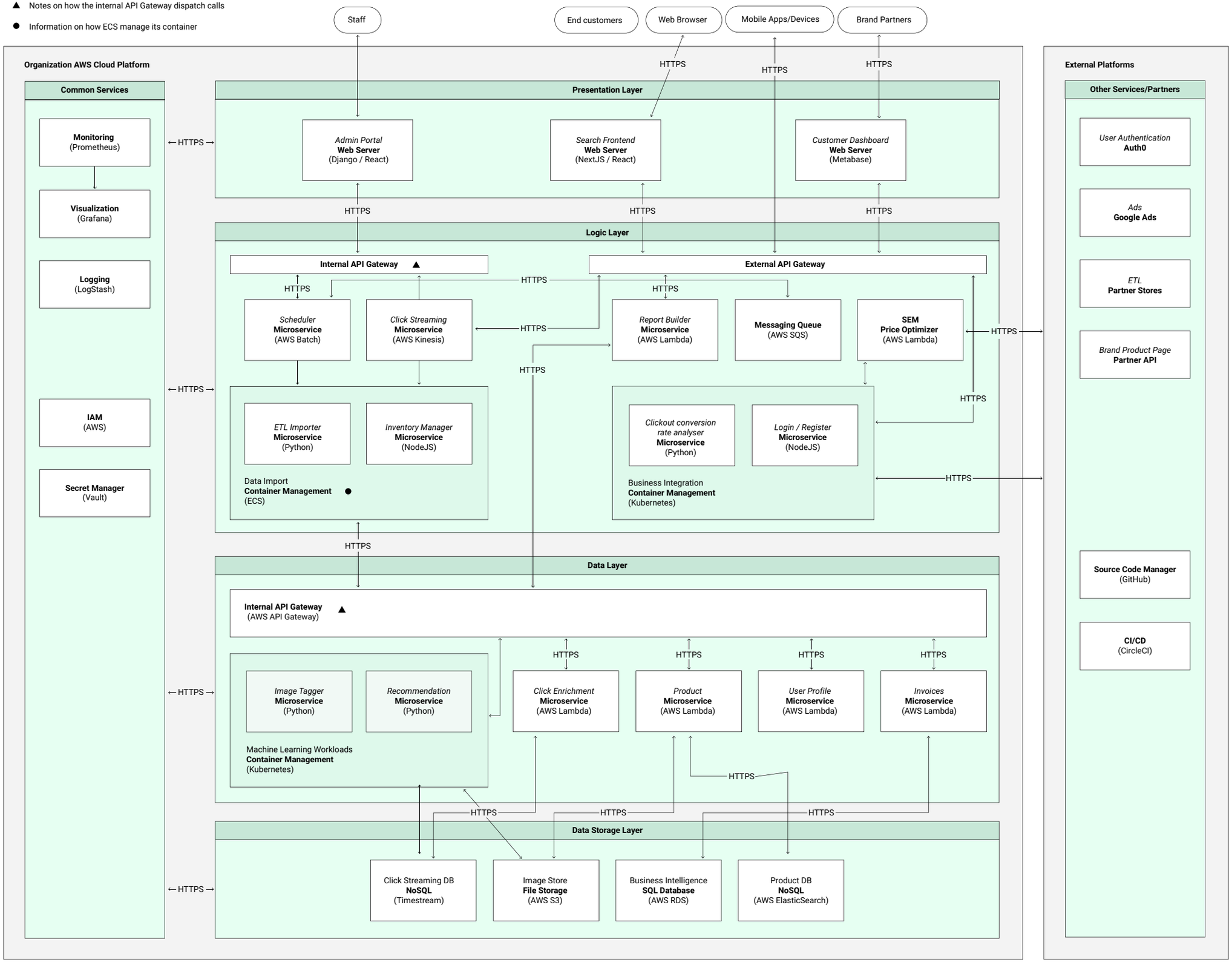
Some of the layers might expose certain APIs (external or internal) so make sure to include those as well. Additionally, make sure to show the other shared services you might have like log management, search engine, message queue, CI/CD, SCM (Source Code Management), and others.

A digestible way to structure this is to connect the components that communicate with each other using arrows and make sure to report which communication protocol is used (HTTPS, gRPC, event message, etc.)

Example of an Application Architecture Diagram:

▲ Notes on how the internal API Gateway dispatch calls

● Information on how ECS manage its container



Infrastructure Architecture Diagram

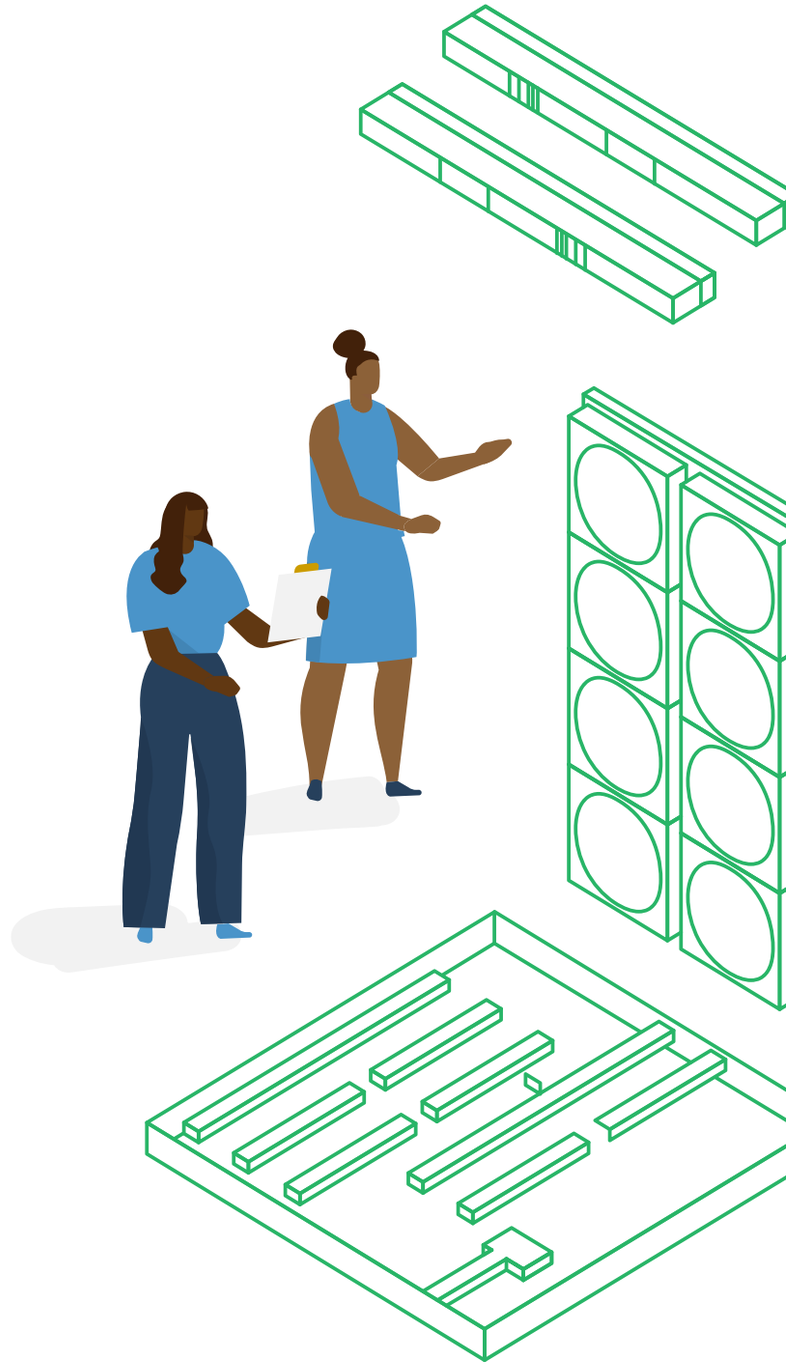
This diagram describes the location of components sitting within your infrastructure as well as how many of them are available. It gives an overview of the scalability of the system and potential Single Points of Failure. This document will also be useful to your team when it comes to sizing the infrastructure appropriately, showcasing fault tolerance and load balancing, as well as easily identifying the root cause of infrastructure-related problems.

In this document, you should represent eventual VPC (Virtual Private Cloud), AZ (Availability Zones), subnets, and instances your logical components belong to, as well as how many of those instances exist. Additionally, you want to represent how those different infrastructure layers are interconnected (Load Balancers, NAT gateways, etc).

Make sure this diagram represents the following components:

- Instance sizes
- Network boundaries
- External system integrations

Example of an Infrastructure Architecture Diagram:



External Users

Internet Gateway

AWS Cloud

Product Search Engine VPC

eu-north-1

private subnet-1c

Apache EC2 (m4.large)

Application Load Balancer

private subnet-1c

API Admin Portal EC2 (m4.large)

Brand Partners Portal EC2 (m4.large)

Product Search Engine VPC

eu-north-1

private subnet-1a

Network Load Balancer

Apache EC2 (m4.large)

Application Load Balancer

private subnet-1b

Data & Business Logic Lambdas

Kubernetes EC2 Cluster

Data Import Services ECS Cluster

Search Frontedl EC2 (m4.large)

private subnet-1b

Machine Learning K8 Cluster

eu-west-2

private subnet-2a

Network Load Balancer

Apache EC2 (m4.large)

Application Load Balancer

private subnet-2b

Data & Business Logic Lambdas

Kubernetes EC2 Cluster

Data Import Services ECS Cluster

Search Frontedl EC2 (m4.large)

private subnet-2c

Products DB RDS (db.t2.xlarge)

VPC Peering

Common Service Platform VPC

eu-west-1

public/private subnet-1a/b/c

Prometheus EC2 Cluster

Grafana EC2 (m4.xlarge)

Logstash EC2 Cluster

Vault EC2 CLuster

CircleCI

API Gateway

Files S3 Bucket

Click DB Timestream

SQS

Backup

Kinesis

Batch

DevOps Architecture Diagram

This diagram represents the flow of processes that come into play when building and deploying your application. It's useful in assisting the team as they review the different build process steps involved across environments, as it helps to streamline and optimize the deployment of your application.

Ultimately, you should be able to represent graphically what is happening in your CI/CD environment. If you use a product like CircleCI, make sure to represent the different orbs that might come into play, such as code metrics, security checks, and image builders.

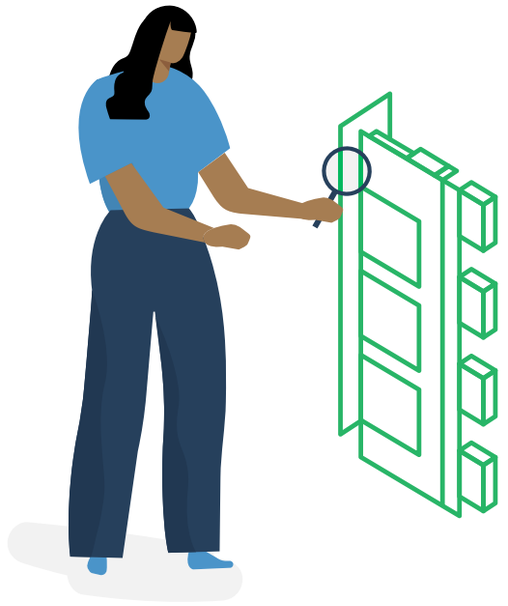
This diagram should illustrate the following:

- Human interventions needed in the flow. While a lot of tasks can be automated, most of the time humans are also part of the process. For example, when the build is promoted from one environment to another, you might want to manually approve the promotion to production only once User Acceptance Testing has been completed by a QA engineer.
- Promotion of the application builds across your development, staging, and production environments (including human processes).
- Notes to explain in detail some of the more complex steps happening in your pipeline.

Data Architecture Diagram

Most businesses these days heavily rely on processing and storing large amounts of data. This diagram should explain how the data is processed, stored, and used across the system.

It's important to periodically review and update this architecture diagram to optimize the cost of processing and storage as your dataset grows.



This diagram should clearly illustrate the following:

- Sizing of the data storage component and data increment rate. This is also useful to review periodically to optimize your storage strategy as you scale.
- How logical components are grouped and their respective boundaries across the system. This is generally segmented into:
    - Data sources: where the data is coming from and how it flows into the system
    - Data processes: how the data is processed, such as ML jobs.
    - Data storage: where the data is physically stored, such as SQL / NoSQL databases, file buckets, and data warehouses.
    - Data backup and data visualization: reporting dashboard, SQL queries, and analyzers.
- How the data flows between different boundaries of the system.

IP agreement signed by the team members

This doesn't always fall under the Tech DD compliance checks but you need to make sure you have signed contracts with every team member, including freelancers. Be very mindful of IP agreements and have them signed before a new hire starts their contract.

Playbook and operation manuals

Any playbooks or documents detailing your processes, such as onboarding documents and processes to deploy code into production are welcome. Auditors are generally reassured when they see that processes are in place, they're documented, and they're followed by the team.

API documentation (if applicable)

Provide any API documentation. A simple Swagger documentation will work.

Open-source libraries list

Pretty much every open-source software library has an accompanying license that specifies the terms of use. Some libraries, despite being open source, cannot be used in a closed-source, commercial

environment, as their license doesn't allow it. Such is the case of the well-known GPL (Gnu Public License), which requires users to open source the code it's touching, making it impractical to use for most commercial applications.

At all times, you should keep an updated list of every open-source library you're using. This is a living document owned by the CTO or VP of Engineering. Ideally, a process should be in place where developers have to submit a request to use a new library and the library must be checked for its license and potential vulnerabilities.

In the case of M&A and advanced Tech DD, you might be subjected to an SCA (Software Composition Analysis), or so-called Black Duck Scan. This is a process where you upload your code to a third-party auditor who then runs scans to detect which open-source libraries are in use, as well as which other libraries they are potentially dependent on.

The auditor is looking for potentially offending library licenses such as the GPL.

Database schema

Database schemas are an important part of the documentation you'll need to provide. Here you want to document all the tables in the database and how they relate to each other, as well as the structure of each table with their associated types. This helps to spot potentially hazardous choices of data types for particular fields, like plain text for passwords.

In the case of a NoSQL database, you might have to provide a sample of the data stored.

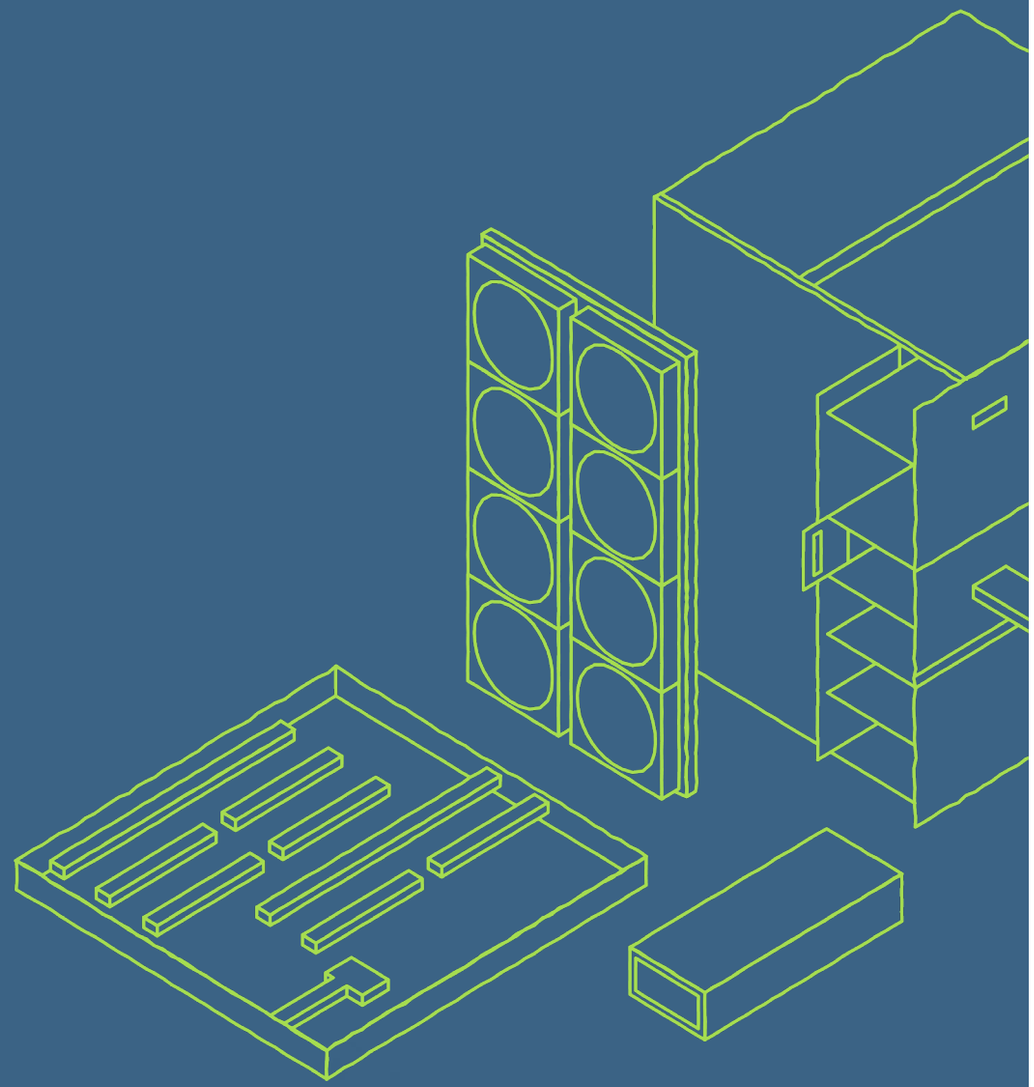
Security audit report

It's wise to have a security audit (so-called pen-testing) performed a few months before the DD process starts. Companies operating in privacy- and security-sensitive industries such as healthcare or fintech submit themselves to, at minimum, a yearly in-depth security audit and should be able to provide historical reports.

Infrastructure and managed services bills for the last 12 months

Providing your infrastructure and managed services bills tells the auditor how you think about optimizing your infrastructure, what costs might creep up when you scale, and how you may have potentially under- or over-scaled your staging and production environment.

General advice



Don’t wing a Tech DD.

The stakes are high — you’re about to either land a round of funding, a new customer, or sell your company. Improvising should not be your game plan here.

You only get one chance to make a good first impression. Keep this in mind from the moment the Tech DD starts and the auditor starts collecting data points. Pay attention to details, from how fast you reply to the first email, to the impression you want to leave on the introduction call. Every detail plays a role for someone whose job it is to paint a picture of the company.

The keys here are: be collaborative, fast, open, and proactive. Never, ever, lie. Auditors generally have a long history of experience behind them and have an uncanny ability to spot inconsistencies. They observe your behavior, tone of voice, and body language as much as what you say. If you don’t know something, don’t pretend that you do. Whenever I have the slightest doubt about something, I’ll always dig much deeper to uncover the truth, even if I have to play “mind tricks,” such as tactical empathy to build trust. If I spot a lie, then my trust is most likely gone and I’ll be reporting this as a red flag to my customer right away.

Confidentiality is key in any type of Tech Due Diligence. This means you’ll want to employ a secure way to exchange information. In any Due Diligence process, using a data room is the de facto way of exchanging documents. Do not exchange documents through emails. If you do,

despite having been advised by the auditor to use the data room, they will most likely assume that your sensitivity to confidentiality is weak and that will start to create doubts.

Be responsive. You are probably overwhelmed with requests from different sides. Those weeks when a Due Diligence is taking place are very hard on the founders, but the timeframe to complete a Due Diligence is generally very short. If you're not responsive enough, you create more pressure on the auditor's end, as they also need time to write a detailed report. Ask the auditor about timeframes, as well as the following questions:

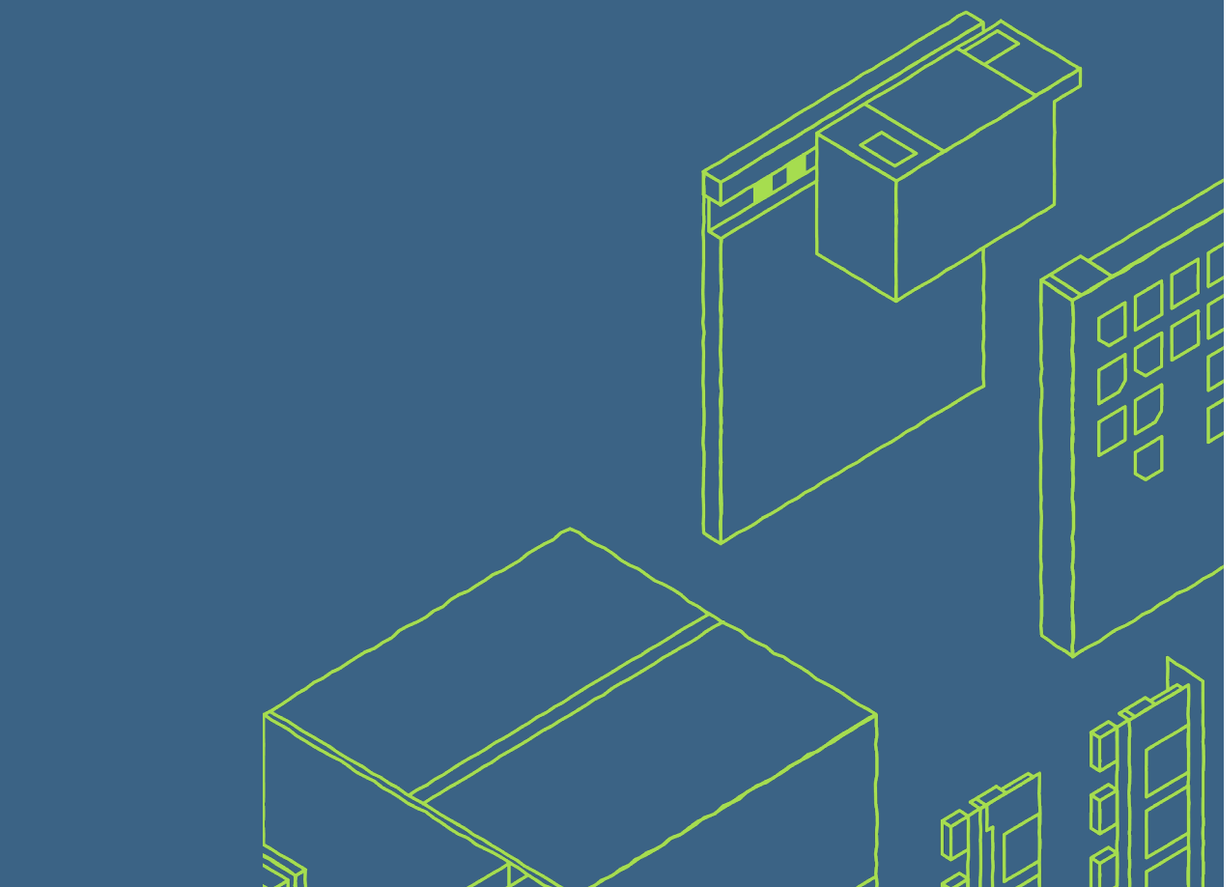
- When am I supposed to produce this document?
- When would be the most convenient for you to perform the interviews?
- Which team members would you like to meet for an interview?

Last but not least, have a plan! Use this guide to produce all the documents you should prepare in advance and have on hand. Don't create these documents at the last minute. In a well-run organization, those documents should always be available and are consistently updated. Not having them generally signals a lack of alignment, consistency, and structure within the organization.

That being said, if you don't have these documents produced yet, you should start working on them as soon as you have any hints that an upcoming Due Diligence might be on the horizon. The good news is that you're generally aware that you're going to raise a round of funding a few months ahead of time.

In the case of an M&A, start producing documents as soon as initial discussions begin — it generally takes a few weeks, to a few months until a Due Diligence kicks off. If you're in an industry where you have large customers and compliance is a key topic, you should get started as fast as possible.

The interview process



What is the auditor looking for?

Different auditors might have different processes but generally, you'll first get an introductory call with the auditor, followed by a "Data Request" document, and then a few rounds of interviews.

During interviews, the auditor will spend a lot of time with the CTO but will also talk to a few engineers on the team.

Before doing any interviews, the auditor will read the data request document entirely and will come back to you with some questions about it, mainly around where they have doubts or elements they don't fully understand in their written form.

During the interviews, they'll mainly focus on a few things:

- Painting a picture of the leadership team and overall engineering culture of the company.
- Spotting potential inconsistencies between what they hear from different team members and what's written in the data request document.
- Understanding the current team composition and how ready the company is to scale on the tech side.
- Understanding the magnitude of the technical debt and the plan to tackle it.

On top of that, they generally like to understand a few other things:

- A company's stance on "buy vs. build."
- How the team thinks about managed services vs self-hosted installations.
- Involvement of engineering with other business units, such as product and marketing.
- Your decision-making process when it comes to technology choices.
- The emotional safety of the team.

Buy vs. build

Here, your auditor is trying to understand the decision-making process when it comes to building things from scratch versus paying for a product that fills the need.

To untangle this in my audits, I like to ask the following questions:

- Is this core to your product?
- Is your decision related to the IPs you're building, or just a necessity to get where you want to be?
- Is there a product on the market that already solves 90% of what you're trying to do?
- How expensive would it be to build and maintain this on your own, versus paying for a commercial product?
- How solid are the companies behind the commercial solutions and what would happen if they withdraw their products from the market?
- Could you easily find an alternative or would you be relegated to build it on your own?

In many cases, if a product exists on the market that solves your need, is affordable, and frees you from spending months of development time, then paying for it will most likely be the right solution for you.

This line of questioning informs me about how you spend time and money to achieve your goals.

Managed services vs. self-hosted:

While 10+ years ago you generally had to install open-source software on your own, such as databases, this has tremendously changed thanks to the increased availability of managed cloud services and SaaS products.

When you deal with complex pieces of software like databases, CI/CD, or messaging queues, one could easily think that installing open-source versions of those products is the cheapest way to go. What

some engineers don't fully realize is that installing, configuring, and maintaining that software also has a cost and requires particular expertise.

Let's take the example of CI/CD. Many products exist on the market that you can install for free, so why would you pay for a commercial product like CircleCI? If you've installed such a solution yourself in the past, you're probably aware that configuring and integrating such a large piece of software can be time-consuming and requires a high level of understanding how the software operates. The time involved in configuration and oversight easily translates into development costs. If you add this to the maintenance costs, such as adding new integrations or monitoring and scaling, in many cases, a commercial product is more cost-effective and provides greater levels of security and stability.

During a Tech DD, the auditor needs to understand how CTOs think about this. The impact on the velocity of the team can be greatly affected if they are too focused on operating pieces of software that are important but not core to the product.

Involvement of engineering with other business departments

As soon as a company grows, it tends to create more silos between business units. Those silos are seen as "separation of concerns" and dictate who's doing what. Marketing does marketing, Engineering does engineering, etc. Unfortunately, this is an overly simplistic view of how a company operates at a deeper level.

I view a company as a complex system, akin to a living organism. In any complex system, the specialization of its parts is just as important as the interactions between those parts. For an organism to survive, each part needs to have two-way communication between them and the rest of the system. In the human body, the heart has an important and specialized function but it also needs to have constant communication with the rest of the body through the nervous and endocrine systems to accommodate new needs (we're running — pump some more blood!).

A growing company shares a lot of those concepts: different parts of the organization have different specializations, but for the organization to function properly there is a need for its parts to communicate efficiently. The more a company grows, the more specialized its many parts will become, and the more communication between those parts will be needed for the organization to function properly and grow.

Making company and team-wide OKRs visible to the entire organization is a good way to communicate and align everyone on the common mission. Additionally, having members of a team periodically spend time with other teams can improve communication and increase empathy. Cross-pollinating knowledge through regular team presentations can also help. For example, organizing a company fair, where each team presents their latest work, asks for advice and feedback, or presents their current challenges.

Breaking the silos between departments by involving engineers in the sales, product, and customer processes can be extremely beneficial. They'll feel closer to the business, develop empathy for the customers, and might even come up with interesting suggestions from their unique perspectives.

How will the auditor assess the communication within the company? You'll likely be asked about how the team is structured and details about the internal communication processes, like regular meetings between departments and how much input engineering gives to product or marketing teams.

While the traditional approach of dividing a company into departments might work for some time, as the company grows, the rigidity of its organizational structure will reduce its agility at an exponential rate. This is why large companies that are extremely inefficient are often disrupted by startup players who move fast.

On the other hand, some large corporations are moving towards smaller, cross-functional teams or so-called "two-pizzas teams," which is the model used at Amazon. The idea behind the two-pizzas team is that every internal team should be small enough that it can be fed with two large pizzas. Teams with a lot of autonomy can operate at a faster velocity than if they have to cross multiple layers of hierarchy and departments to get things done.

> ***Any organization that designs a system (defined broadly) will produce a design whose structure is a copy of the organization's communication structure.***
>
> *— Melvin E. Conway*

To put this quote in simple terms, the software architecture you will produce is highly dependent on the structure of your organization. In very concrete terms, a cross-functional organizational structure will generally result in a microservice architecture. A team and the code they produce benefit from more autonomy. Scaling becomes more approachable as the chance of stepping on each other's toes or waiting for a chain of approval to be completed becomes much easier.

Does that mean you should immediately adopt microservices and move to cross-functional teams? Certainly not. It all depends on the size of the company. There are certain benefits in having a more monolithic structure (both in terms of teams and software architecture) if you're small. Starting a microservices architecture when your organization is still very young might lead to a lot of overhead, and premature scaling is one of the highest causes of failure for a startup.

Discussing technical debt

The term "technical debt" refers to the shortcuts that engineers intentionally take to prioritize the speed of execution.

Technical debt shares a lot of commonalities with financial debt: you take a credit that you're supposed to repay later. If you wait too long to repay your debt and keep accumulating new debt, it will eventually get out of control and will become harder and harder to repay.

In the software engineering world, the debt's interests are manifesting themselves in the form of buggy software, loss of confidence in the source code, and reduction in velocity. You initially took the debt to go faster but at some point, this debt comes back and hinders your ability to deliver quickly and reliably.

Managing technical debt is a question of balance but unfortunately, many companies lack a systematic approach to keeping it manageable.

If you answer the auditor by saying that you don't have any technical debt, the auditor will think that either you're being deceptive, unaware of the debt you do have, or simply that you're sacrificing speed of development. There are a few exceptions to this. Perhaps you're working on a critical system where any technical debt will have heavy consequences on the end product and the customers using it. This concerns products where safety is an issue or where updates are difficult, like in embedded systems.

The best approach to tackling technical debt is to actively keep track of it and incorporate its resolution into your sprints. You can do this by dedicating a percentage of the team's time to code cleaning, refactoring, and ensuring test coverage, among other things.

The auditor will generally ask you questions such as:

- How much technical debt do you believe exists up to this point?
- How do you log technical debt?
- How much time is dedicated to its resolution per sprint?
- How do you prioritize what should be fixed first?
- What is its evolution rate?
- Do you create more debt than you tackle?

Technical debt hides in many different places such as:

- Architecture choices
- Test coverage
- Portions of code
- Database structure
- Technology stack

Not all technical debt is equal. While it might be okay to have tech debt on internal tools or dashboards, it might be more problematic if your core architecture is plagued with it. An auditor might not be too concerned about the lack of unit tests on the frontend but will expect the backend code to be well-covered with it. You should follow the same thinking when it comes to prioritizing your tech debt resolution: start with the most critical part of your system first and move upward to less critical parts.

Decision-making process

The way decisions are made has a huge impact on the success of the company. Here your process may vary from gut feelings to data-driven decisions. You might lead through consensus or prefer to go with what seems to be the best idea despite what other team members think.

As an auditor, I need to understand how you think about making decisions. I've seen CTOs give a very large degree of autonomy to the team with very little oversight, while others get closer to micro-management.

In extreme cases, I've seen CTOs who are so removed from the decision process that the team sometimes goes wild with their technical choices, implementing niche languages, or making a poor judgment on some key aspects of the architecture. You want to strike a balance and this is what you'll be judged upon.

What I generally like to hear regarding a decision-making process are:

- We listen to everyone's voice, but the best idea wins.
- We inform our gut feeling with data and constantly refine it through retrospectives.
- I'm direct but caring. I don't fear conflicts but I act with empathy.
- I oversee any major architectural decision and provide my input.
- I act as a coach for my team and I'm there to unblock people.
- We have a "no-blame culture." We look at the processes and see what went wrong.

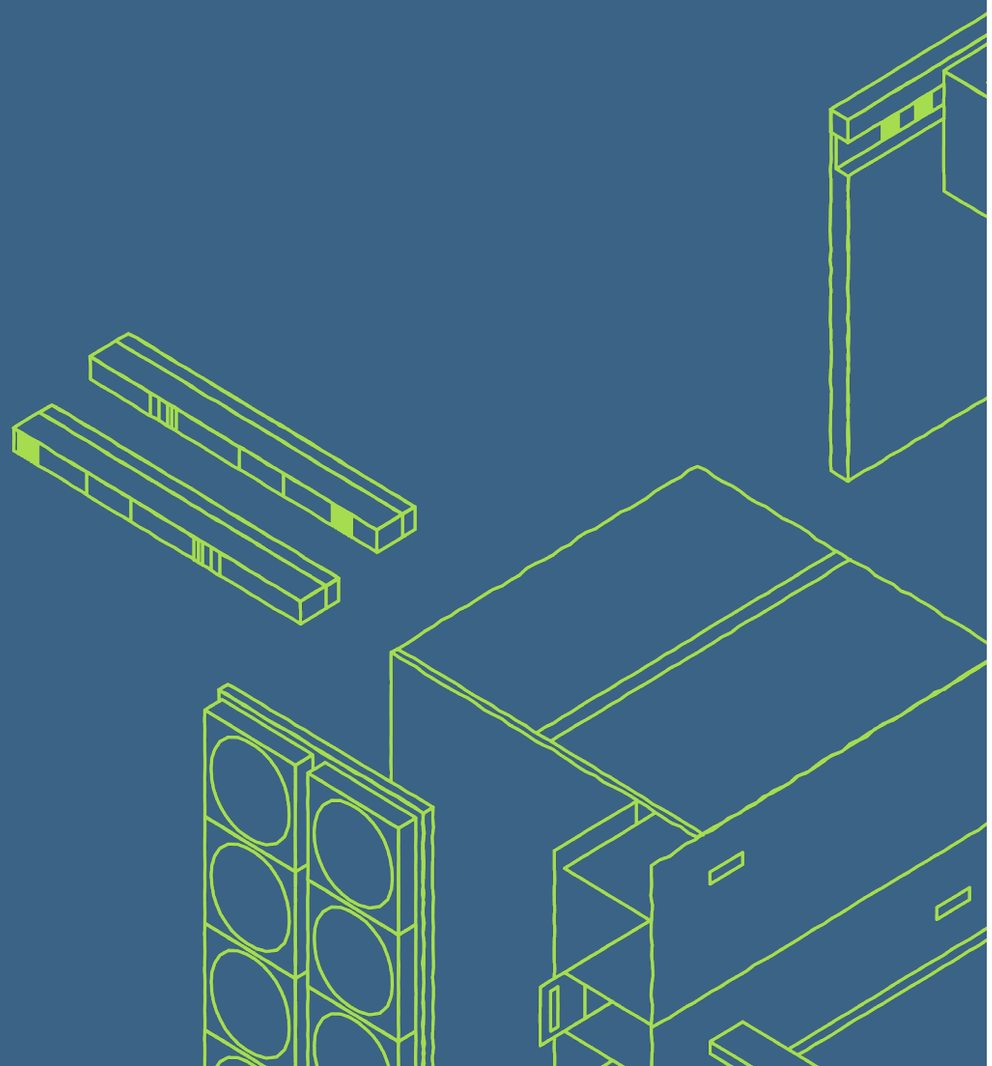
Psychological safety

A Google study in 2016 showed that the best-performing teams are the ones with a high level of psychological safety. This is one aspect I try to uncover during my Tech DD interviews by building a high level of trust with the interviewee. My goal is to make them feel comfortable opening up, talking about how they manage conflicts, and how they feel about their job and personal growth.

From those questions, I can infer the overall empathy within the team. If I hear that there are no conflicts on the team, I always get extremely suspicious. The same suspicion goes if they are being very succinct in their answers — this is generally a way to signal they'd rather not talk about the topic.

While this may not seem related to technology, assessing the overall feeling of the team regarding their managers, C-suite, and general strategy is important, as those elements have a measurable effect on productivity.

The data request document



What is this document?

The data request document is very important for the auditor to understand, in-depth, the technological underpinnings of what has been built to date, as well as how you got there.

This document is generally segmented into different categories such as Team, Architecture, and Infrastructure. Some of the questions might appear to be quite similar but they each serve a purpose. Auditors tend to cross-reference the answers to spot potential inconsistencies.

Even though your timeframe to return this document is usually limited to a few days, it is important to provide detailed answers and avoid replying with a mere “yes” or “no.” For example, you may be asked if you’re actively practicing Tech DD. If you do, the auditor most likely wants to know some details about how you do it. If you don’t, it’s important to write down why and explain your plan moving forward.

Another thing to note about the data request document is that some questions are more important than others for the auditors. But since you don’t know which those are, you should treat all questions with the same degree of importance. Different auditors have different pet peeves and biases, so make sure you balance the level of effort you put into answering their questions.

Architecture

Logical resiliency and failover analysis

An auditor will want to see at least two architecture diagrams: one to get an overall idea of the main components of the system and how they interact; and a second one that provides granular detail of the services involved and the flow of messages within your architecture. You should be able to run the auditor through the diagrams and preemptively answer questions such as: what are the single points of failure? What happens if this load balancer or this component goes down?

Again, be extremely transparent about the current state of things, but make sure to also have a plan for how to mitigate single points of failure. In my experience, nearly every system has single points of failure if you dig deep enough.

Scalability

Depending on your architecture, you might be thinking about scalability in different ways. If you're running a monolithic application, it's generally about vertical scaling which has its limitations, like reaching the single machine capacity. You'll then be asked about how the system will behave with 10x more users and when you plan to introduce horizontal scaling.

It's generally useful to perform load-testing to assess the current capabilities of the system and know where it will break down.

Beyond the hypothetical 10x increase in load, the auditor wants to understand where company growth might lead to trouble with the current architecture, and if your plan to address it matches the product growth strategy. Generally, when your company is subjected to a Tech DD, an expectation of rapid growth is implied. The question that the auditor is trying to answer is: given this expected growth, how ready is the tech architecture to accommodate it and how long will it take to get there?

In my practice, I've often seen companies accumulate a lot of tech debt to get where they are, possibly through many iterations, pivots, and feature prioritizations. While the company might be investable from a business perspective, the main question is: can the technology accommodate growth in a scenario where they will be acquiring users at a much faster pace? If the team has to dedicate 6 months to rebuild weak foundations, the impact on the business might be so massive that a large part of the investment will go into fixing shaky parts, rather than growing the user base, which is not the desired outcome.

Analytics

Usage analytics

Application usage metrics are generally an important part of the KPIs of most companies. Having no analytics is akin to navigating a ship without a compass or GPS. You have no idea where you're going.

You're expected to have a good analytics platform in place with a solid dashboard allowing you to see the most important metrics at a glance, as well as the ability to dive deeper into particular metrics. Don't implement your own analytics platform as many existing platforms are available as SaaS products.

If you have a B2C product, you need to make sure to track:

- Funnel conversions: How do your users behave through the different steps of a registration process. Where are you losing users?
- Cohort analysis: Since your product is constantly evolving, being able to track the behavior of different groups and corresponding to different releases is important to better understand how your changes affect other dimensions of your analytics such as churn and duration of session.

Crash analytics

If you're developing a mobile app, ensure you're able to track crashes of your application. Services such as Firebase Crashlytics can be a big help here. Crash analytics allow you to get a better understanding of how many times your app crashes and upon which particular event, for example, UI, network, or logic.

Beyond potential questions about which metrics you track or how your conversion funnels look, you might also be asked to provide screenshots of your analytics dashboards.

Infrastructure

Logical resiliency or failover analysis

Depending on whether your infrastructure is deployed on-premise, in the cloud, or as a hybrid model, you'll have to cater to different constraints.

The auditor wants to understand how resilient and scalable your infrastructure is. This information should generally be made available through the Infrastructure Diagram by describing the redundancy of the system as well as the failover techniques in place.

Some of the important topics you'll have to cover here are:

- Provisioning automation and infrastructure-as-code
- Logging management: what do you log, how do you transport those logs and aggregate them, and how do you analyze them?
- Monitoring infrastructure: what did you put in place to monitor your infrastructure and at which levels?

You might also get some of these questions:

- Are you using several availability zones?
- How long does it take to provision a new instance in your infrastructure?
- How easily can you spin up a new server instance and provision it?
- How much monitoring is in place?
- Are you solely using your cloud provider's dashboard or any other third-party dashboards?
- Do you have any predictive event analysis to spot problems before they occur?
- What's your logging strategy? Do you mine logs for particular events or patterns?
- What's your scaling strategy? How do you know when you need to increase capacity?

Products you might find useful to integrate into your CircleCI pipeline:

Terraform

HashiCorp enables organizations to adopt consistent workflows to provision, secure, connect, and run any infrastructure for any app.

Accurics

Accurics codifies security into development pipelines to detect and fix cloud infrastructure risks. It maintains a secure posture in runtime by mitigating risks from infrastructure changes.

Coralogix

Coralogix is a machine learning-based logging platform that helps software companies manage their log data, ensure quality version release, and automatically identify problems in their production.

Bridgecrew

Bridgecrew helps teams secure their cloud. By leveraging security-as-code and automation, Bridgecrew identifies and fixes cloud infrastructure misconfigurations in run-time and build-time.

LogDNA

LogDNA's log management platform provides deep insight into development and production environments. LogDNA allows teams to ingest, aggregate, and view log data regardless of data residency and infrastructure.

Cost analysis

When it comes to cost analysis of the infrastructure, the auditor wants to assess whether your infrastructure is under or over-scaled. There's a balance to be achieved between being cost-effective and providing a large amount of fail-over.

Data modeling, processing, and storage

As an auditor, I like to spend quite a bit of time on the data management aspects of the companies I'm screening for a few reasons:

- The lack of query optimizations in your database is often the root cause of bad performances.
- There are compliance aspects of data management when it comes to General Data Protection Regulation (GDPR) and the California Consumer Privacy Act (CCPA), plus HIPAA compliance if you're in the health tech field.
- From a security perspective, data management is one of the most vulnerable parts of the system as it can hold a trove of sensitive information.
- From a disaster recovery perspective, things have to be in order to successfully pass a Tech DD.
- If you're processing and storing large amounts of data, the cost associated with it will be a point of focus as well, as it can be expensive to properly manage this.

For this portion, a data architecture diagram will be key for the auditor to understand everything from relational mapping, to data transformation, down to data storage. Refer to the data architecture diagram mentioned earlier for more information about this.

The auditor wants to understand the overall architecture of the database: Is the application write and/or read-intensive? Do you use a primary/replica configuration? Are you using any delayed replica for additional safety in case of database corruption? Are you using any Object Relationship Mappers (ORM)? If so, how much do you know about it, and do you keep an eye on performance? Are you denormalizing your database? If so, why? Do you cache query results?

- Describe all the data stores you currently have in place
- Are your data stores managed services or did you install your own?
- What's the backup strategy for each data storage component?
- Do you track poor performance queries on your database?

- Do you do any machine learning? What's the current strategy for it?
- If you're using any non-mainstream database such as PostgreSQL, explain the rationale behind this choice.
- Who has access to the database?

A Product to consider adding to your CI/CD pipeline:

Nightfall
The first cloud-native data loss prevention platform that uses machine learning to discover, classify, and protect sensitive data.

Application performance management

Early on in my CTO career, managing performances was not on top of the priority list. In fact, users started to complain about how slow the app would load, or how unresponsive the UI could be. This is something you shouldn't let happen in your company.

It's important to measure the performance of the overall application but also to instrument your code to figure out where the performance degradation might be originating from. It can be a poorly written piece of logic code, a service that is not properly load-balanced, a slow database query, among other possible reasons.

Many tools and products in the application performance management space already exist — don't reinvent the wheel but do search for the best tools. Most likely you'll end up implementing tools like Pingdom to measure performance from outside of the organization as well as products like New Relic, Splunk, or EverSQL, which will give you insights on different layers including mobile, backend, or database queries.

Here are a few examples of questions you might be asked in the data request document:

- How is the performance of the application monitored and tested at a high level and a user level?
- Which performance metrics are tracked in the frontend and backend applications?
- Is possible performance degradation tracked after code changes or releases, and if so, how?
- How do you deal with user-reported performance issues?

A product to consider integrating into your CI/CD pipeline:

Blackfire
Blackfire aggregates profile data collected by the probe from your application engine before sending it to servers.

Product management

While building a company, the technology itself is generally not the end goal. The technology is an enabler for the product you want to put on the market, and the product enables you to build your business. That's why the technology must align with the product and the business, all coordinated by stable processes, constant feedback, and iterations.

Here the auditor wants to understand how you build the product and assess the processes in place.

The auditor might also look into the lifecycle from a feature idea to its rollout in production, from a process perspective. Documenting processes is a very important part of product management. The more unknowns there are, the more iteration will be needed, sometimes to the point where the implementation does not match the initial idea.

In my practice, I would generally ask for screenshots of your ticket tracker (such as Asana or Jira) and any documentation related to the product and the process of development associated with it. Just knowing how many meetings there are and their respective format can be a strong indicator of how well the team thinks about product development.

The decision-making process as mentioned above will be scrutinized here. Expect questions such as:

- How does management come up with the new features to be implemented?
- Are they using a data-driven approach?
- How do they assess product-market fit?
- What are the KPIs of the product?
- At which point of this process does the engineering team get involved?
- Are developers providing customers support or assistance?
- How do you document the specifications of the product and how often do you revisit it?
- What kind of agile process (if any) is in place?
- How long are the sprints? Do they take into account tech debt management?
- Are you using any feature flags to roll out new features progressively to your customers?

Products to consider integrating into your CI/CD pipeline:

Jira
Jira is a family of products built to help all types of teams manage their work. Jira offers several products and deployment options that are purpose-built for Software, IT, Business, Ops teams, and more.

ConfigCat
ConfigCat is a feature flag and configuration management platform that provides a single place to manage frontend, backend, mobile, and desktop apps.

LaunchDarkly
LaunchDarkly is a feature management platform that software teams use to build software with less risk. Teams can control their entire feature lifecycles from concept, to launch, to value.

Development process

The software development process is a fundamental part of building a product, yet many startups don't put too much effort into it, as it's generally an investment you make in the future.

The development process encompasses testing, source code management, and coding guidelines.

Testing: unit, integration, UAT

If we look at software as a complex system, it's important to be able to verify that a change at one point in the source code won't produce unintended effects at another point. In the mid to long term, the absence of testing will result in more and more bugs, and will eventually lead to the team losing trust in the codebase. As a result, the quality of the product will decrease and more importantly, the velocity of the team will be greatly affected.

Test-driven development (TDD) is the practice of creating tests that expose the desired behavior before writing the actual code. Most startups, at their inception, will go without testing to achieve greater velocity. But at some point, when the codebase starts to increase in complexity, the situation will reverse, as the codebase will become more and more unstable. This becomes technical debt.

To ensure a certain level of stability, the code has to be covered by tests at different levels:

- Unit tests
- Integrations tests
- User acceptance tests

Even though tests are written manually, the way they are executed should be automated, ideally through your CI/CD pipeline.

Products that can be integrated into CircleCI to manage healthy test coverage:

Codecov
Codecov helps teams improve code review workflow and quality. Use Codecov's integrated tools to group, merge, archive, and compare coverage reports.

Happo
Happo is a cross-browser screenshot testing tool. It helps prevent visual regressions before they reach production and lets you see how a change visually affects the UI.

Cypress
Cypress is a front-end automated testing tool, created for the modern web. Cypress provides more reliable testing for anything that runs in a browser.

Coveralls
Coverage is posted to Coveralls for analysis and tracking, and adds coverage change in a comment on PR builds.

Source code management

You're likely already using a Source Code Management (SCM) tool such as GitHub or Bitbucket. If you're not, you should immediately stop reading this and start using one!

One thing the auditor might look at is your strategy for organizing code. Are you using a single repository (mono-repo) or several (multi-repo)? While there is no right or wrong when it comes to using one or the other (or anything in between), you need to have clear arguments on why you chose to go that way.

A mono-repo can seem counterintuitive for large projects but companies such as Google use them because they considerably simplify many aspects of the CI/CD pipeline, and help you avoid getting lost in a sea of hundreds of repositories.

Multi-repos on the other hand, have some advantages when it comes to potentially sharing part of your code with external partners or developers because you can give granular access to a determined portion of the code.

When you go multi-repo, a product like CircleCI, which integrates with GitHub and Bitbucket, becomes key in managing the complexity of the build process.

Make sure you have a good rationale, no matter your strategy.

Coding guidelines

To keep a certain consistency in the codebase and ensure a high level of readability, having clear coding guidelines that are actively enforced is essential. Again, your best bet for this is a CI/CD pipeline to automate the process of checking the code produced against the coding guidelines you've set up.

Code security analysis

Writing secure code is an art. Many products plug into your CI/CD pipeline that will help uncover potential security issues by performing static code analysis.

Products that integrate with your CircleCI pipeline:

Sken.ai
Sken.ai is an application security testing product that offers comprehensive testing for DevOps without the need for security expertise.

Code quality analysis and tech debt

We all write suboptimal code from time to time. Static code analysis tools will help you spot code that may have to be refactored in the future and keep track of it.

In the data request document, expect questions such as:

- How much of the code is covered by unit tests, integration tests, and UAT. Detail this coverage by grouping them in logical units like backend, mobile, frontend, and ML jobs?
- Which tools are you using in your CI/CD environment to check code quality and adherence to coding guidelines?
- What steps have you taken to make sure the code written is secure enough?
- Do you automatically track tech debt and keep a log of it? How?
- What are the different steps in the build process that need to pass before the code can be promoted to the next environment, for example, from staging to production?

Depending on the depth of the audit, a third-party company that specializes in code quality assessment might be mandated to look at your code repository. As an auditor, I generally won't ask to see any code because I can derive enough insight from the data request document and interviews.

A product you can integrate into your CI/CD pipeline:

CodeScene
CodeScene is a visualization tool that uses predictive analytics to prioritize technical debt, detect code at risk for defects, and uncover team productivity bottlenecks.

Team

As a CTO, your day-to-day responsibilities usually evolve. While you might still be coding, more effort will be put into building and leading an engineering team.

The auditor will most likely interview several members of your team to better understand the dynamics of the whole engineering department and potential issues in management. Interviewing several engineers gives good insight into the engineering culture and the psychological safety of the team, which is an important part of a productive environment.

Here are the type of questions you might be asked:

- Hiring funnel and process: what does your hiring process look like? How long does it take to hire an engineer and onboard them? Do you have onboarding documents? How long does it take for the recruit to commit their first line of code?
- Is there a balance between the skills and experience of the team members? How many senior engineers are on the team?
- Is the team provided with the correct equipment to carry out their work?
- Are the engineers provided with a personal development budget to buy books, online courses, or go to conferences?
- How much is personal growth taken into account by managers?
- How involved is the engineering team in helping define the product?
- Are engineers doing customer support from time to time?
- Give an overview of the meetings organized every week. How much time does engineering spend in meetings on average per week?

DevOps / SRE

The DevOps architecture diagram will be key in helping the auditor get the overall picture of your DevOps activities. It should represent all the steps involved in getting code from the development environment to production, including how the code is deployed.

In a well-thought-out DevOps process, a lot of automation should be in place to ensure that the code produced is thoroughly tested, checked for quality and security, and deployed. In my opinion, CI/CD is mandatory even if you're the sole developer working on a project. Humans are notoriously bad at repetitive tasks, and that's where CI/CD comes into place. Automation not only helps prevent costly human errors, it tremendously reduces the time required to bring good quality code to production.

Here are the types of questions you should expect in the data request document if they are not already covered in your DevOps architecture diagram.

- Which CI/CD environment are you using?
- What third-party software is part of your CI/CD environment? What do they do?
- What are the human steps required in the build process?
- How is the final code bundled and deployed to production servers?
- How do you track potential deployment issues?
- Do you have a rollback strategy in case of a major issue with a new release that was just deployed into production?

What if you don't have a DevOps position within your engineering team? While some companies will proudly endorse being "NoOps," this will generally raise eyebrows from an auditor. In any well-run organization, automation and care of processes must be implemented and maintained. Even with current state-of-the-art technologies in place, it takes human brainpower to make it performant and cohesive.

In a microservices architecture, DevOps becomes even more important as the number of moving pieces increases and the need to continuously deploy them independently increases.

Products to integrate into your CI/CD pipeline to make your life easier:

Rollbar
Rollbar helps engineering teams deploy more often and with confidence. Use Rollbar to detect all production errors in real-time and debug them before users notice.

Rafay
Rafay provides a SaaS-based platform that lets you create, deploy, operate, monitor, and upgrade Kubernetes clusters & K8s resident apps across multi-clouds and environments.

realMethods
realMethods makes it easy to get running on CircleCI, by generating MVP quality applications for all major tech stacks committed to your Git repo with a ready-to-deploy CircleCI config file.

DeployHub
DeployHub is a central catalog of deployable objects where developers can publish to and deploy from. DeployHub is useful for both agile practices and modern microservices architectures.

Documentation

At an early stage, companies tend to have very little overhead and optimize for speed rather than process. But as a company matures and the team grows, more and more processes have to be put in place. Oral communication does not scale and documentation should take precedence. Documentation should ideally be centralized in one place, indexed properly, and searchable.

An auditor will usually ask you to provide product documentation, development processes documentation, and they might even look at comments on your code. Documentation should be time stamped and updated regularly. Teams should be trained on the importance of writing good documentation and most likely, you'll have to document that as well.

A product that can help keep your documentation organized:

Confluence
Confluence lets you build, organize, and collaborate on work in one place from anywhere.

Security and data protection

Security is a sensitive topic. In some Tech DDs, security will become a separate audit, usually through a penetration testing company.

The auditor will pay attention to your security "mindset" during the call. Be particularly careful when you share your screen — the auditor will look at things like your password lengths when you log in to internal resources or notifications showing that your browser or OS is not up-to-date.

The same goes for showing potentially sensitive information and non-masked data during the call. If you're being asked to show some code through screen sharing, the auditor will attentively watch whether you're using Two-Factor Authentication (2FA) and the size of the password you're entering.

If your infrastructure is hosted on-premise, it should have sufficient physical and operational security as well.

To understand your security practices, the auditor will check the following:

- How are privileged resources managed? Who has access to what? What are the fundamental principles you apply to grant or deny access? As a rule of thumb, you should apply the principle of least privilege, which is the idea that any user, program, or process should have only the bare minimum privileges necessary to perform its function.
- Do production servers have SSH access? What does it take to log onto them? Are they IP-restricted over VPN?
- How do you store secrets? When a process needs access to a privileged resource, where can it find the needed information to connect? The best practice here is to store secrets in a vault. Most cloud providers offer vaults, so there is no longer a reason to use environment variables.
- Are your internal communication channels encrypted?
- How sensitive is the data stored on your servers? How easy would it be for a malicious actor who gained privileged access to your database to extract sensitive information such as passwords or personally identifiable information?
- How is sensitive data stored?
- Do you encrypt your backups in case they end up in the wrong hands?
- Do you use encryption at rest and in transit?
- Are you using role-based access control (RBAC) and with which level of granularity?
- Are you using any Fuzzing products to limit 0-day attacks?
- Are you sanitizing inputs on mobile and web interfaces?
- Authentication: It is generally recommended to use a 3rd Party such as Auth0 instead of building your own. While it's common for startups to use their own authentication scheme, it also creates a larger surface of attack for malicious actors.
- How often are you doing security audits?

Products to help with security management that integrate into your CI/CD pipeline:

Snyk
Snyk is a security solution platform that finds and fixes vulnerabilities and license violations in open source dependencies and containers.

Governance

Governance and compliance topics are high on an auditor's list, as they have legal implications that should not be underestimated.

Open-source licenses

You should always keep an updated list of every library in use, along with their licenses. You should also have a process in place requiring engineers to report new libraries being used and be trained on which particular types of licenses to avoid, such as GPL, because they can't be used commercially (unless you want your code to be open-source as well).

Don't consider this topic lightly. In some cases, you might be subjected to a Software Composition Analysis (SCA) scan, also known as a "Black Duck" scan, where you'll have to upload your entire code repository to an air-gapped server for it to be analyzed by software that will check all the libraries in use, as well as their dependencies, to produce a full report.

A CI/CD platform can help by running SCA scripts systematically on each build thanks to the integration with WhiteSource. Snyk can be used for this as well.

GDPR / CCPA compliance

If you do any business in the EU, GDPR will apply, even if your company is based abroad but some of your customers are in the EU. If you're incorporated or have customers in California, CCPA will apply.

When it comes to GDPR compliance, a specialized lawyer will likely get involved and collaborate on the technical aspects with the tech auditor. Depending on your country of incorporation and your industry, some specific laws may apply.

Of particular importance when considering GDPR and CCPA is how your store, process, and sell your customers' personal information. If you're using a third-party provider to store certain information about your customers, such as authentication or authorization providers, you'll need to make sure they provide full compliance with the law as well, such as "The right to be forgotten," meaning that you should be able to prove that you will delete customer data upon their written request.

GDPR is on the list of show-stoppers if you're not compliant and you could also face a hefty fine.

HIPAA compliance

This only concerns you if your company is in the healthcare industry. Since HIPAA is a complex topic, a lawyer will be involved here and working in tandem with the tech auditor.

Not respecting HIPAA compliance could tank your investment or acquisition deal as well as bring you in front of a court.

IT security

When it comes to security, it's quite important to make sure that your internal IT security is strong, as it can be the easiest point of entry for a potential attacker to infiltrate your system.

Here, an auditor will look at things such as:

- Security of the communications between internal and external IT systems. Are you using a VPN? If so, do you have a backup VPN in case the first one is down?
- Security of employee computers: Do you have a fleet management system? Can you erase a disk remotely if a computer has been compromised or lost? Are disks encrypted?
- Termination protocol for employees leaving: Do you have an automated system in place, or a written procedure enabling you to revoke an employee's access from all the resources they normally have access to? Does it take less than 15 minutes?

Public website

While the public website tends to fall under the responsibility of the marketing team or an agency, some technical aspects of it might make a difference for the business. SEO is a key aspect of building a strong online presence, which enables your business to reach customers organically. I'm always surprised to see many founders investing massive amounts of money in performance marketing, while their website shows a poor SEO ranking.

Start with Google PageSpeed Insights. Google attaches particular importance to the performance of your website which influences your SEO ranking. If your score is below 90, look at the suggestions this tool makes — many of them are low-hanging fruit for a frontend engineer to fix.

Most likely you'll be asked to provide a list of keywords you optimized for, and

depending on how critical the website is for the business, the auditor might dig deeper into how this has been implemented, using free SEO tools.

The questions below might come up in the data request document:

- What is your SEO Strategy?
- How many backlinks does your domain have?
- What are the targeted keywords?
- Provide analytics about your organic traffic as a screenshot.

Be aware that the auditor might use tools to verify the trustworthiness of your answers and challenge you if they find something different on their side.

Make sure to have web analytics in place. It will reflect poorly on you if you don't have Google Analytics in place, even though your public website might be a simple landing page advertising the company's services.

Products you can integrate into your CI/CD pipeline if you're using WordPress:

Pantheon
Pantheon is a WebOps platform for Drupal and WordPress that helps organizations get more value from their sites through hosting, automated workflows, and professional tools for teams.

Disaster recovery

During the Tech DD process, disaster recovery is a very scrutinized topic. At the same time, it's a topic that is often underestimated by CTOs as it covers worst-case scenarios, like your data center burning down. Even if these events are rare, they sometimes happen and nothing should be left to chance.

The keywords here are backup and automation.

You're expected to produce a disaster recovery plan covering unlikely scenarios that could potentially put the company out of business. What happens if you lose one of your availability zones? What if the database server disk crashes? What if you're the victim of a DDOS attack? What if a malicious actor gets privileged access to your system?

Here is a list of checkpoints to have in mind:

- Do you have automated and scheduled backups for everything? Are those backups stored in the same location or delocalized?
- How fast can you recover if you'd have to rebuild the entire infrastructure? Do you have automation in place when it comes to provisioning servers, redeploying code, and recreating your database from backups?
- Do you automatically test your backups to make sure you recover from them?

The bottom line here is that you should be able to reliably rebuild your whole infrastructure from the ground up in less than a few hours. The best way to get ready for this is to pretend that your main data center just disappeared and simulate a full restoration of your system. Assess what the suboptimal processes are and iterate until you get to the desired result. Document everything and revisit this document every quarter to see if anything has changed.

It's wise to test the resilience of your system by practicing chaos engineering. Pioneered at Netflix in 2011, chaos engineering relies on a piece of software (called Chaos Monkey at Netflix), which will simulate incidents by randomly killing server instances. By now there are tons of variations of this idea, ranging from orchestrating failure in a Kubernetes cluster (Chaos Mesh) but to dropping an entire Availability Zone (Chaos Gorilla) or even simulating the disappearance of an entire datacenter (Facebook Storm).

Product support

An important part of the customer relationship is product support. Here there are several elements to take into account.

Channels

Not so long ago, product support was as easy as providing a phone number or an email address to respond to your customer's feedback. Nowadays, any organization must adopt a multi-channel approach, as feedback comes from many different platforms, including social media and website forms. Ideally, you're tracking hashtags, @replies, and Direct Messages, and responding to your customers by pointing them to your customer desk system. Many SaaS products exist, like Zendesk, Intercom, or UserVoice, that can assist here.

The goal is not only to capture potential bugs but also to gather feedback on possible improvements. Once a ticket is created inside the system, it's much easier to track and keep the communication flow going with the customer or user.

Incident response flow

The incident response flow is how a ticket gets escalated to the tech team. There are many levels of severity but you want to be able to escalate an item that's been reported by several users such as the service being down or a payment system not working. Hopefully, your monitoring system will capture the incident before your users but in both cases, you need to define the process by which an alarm is triggered and some team members are contacted automatically to resolve the issue.

Response time (SLA)

Depending on the nature of your business, your mileage will vary here. B2B businesses should define SLA (Service Level Agreement) by determining how long they need to respond to an open ticket.

You should keep a log of those statistics and it's a good idea to set a business KPI around the response time to reported incidents.

When it comes to B2C companies, the incident response is usually more flexible, but it's a good business practice to be as responsive as possible.

Uptime and downtime (SLA)

For most businesses, a downtime of the service means a loss in revenue or a loss of customers. Make sure to keep detailed records of downtimes and set a KPI around that as well.

For B2B companies, and sometimes even B2C, a downtime might contractually mean that you'll have to pay a fee for the interruption of business as it could severely impact the business of your customer as well. Let's say you're building a payment gateway and your system is down. That might impact several thousand customers who can't charge their customers and end up losing revenue.

Make sure to negotiate SLA properly as the consequences on your business might be dramatic. There is a huge difference between providing an uptime guarantee of 99.9% of the time versus a 99.99999% which could translate into building your system for high availability. Building a highly available system means having a lot of failover and redundancy in place, resulting in a much higher cost of infrastructure. Ensure that your SLAs match your capability (both in terms of human resources and financial resources) of running a highly available system.

It's also important to do a postmortem with the team whenever the system or part of it is down, regardless of the time it was down for.

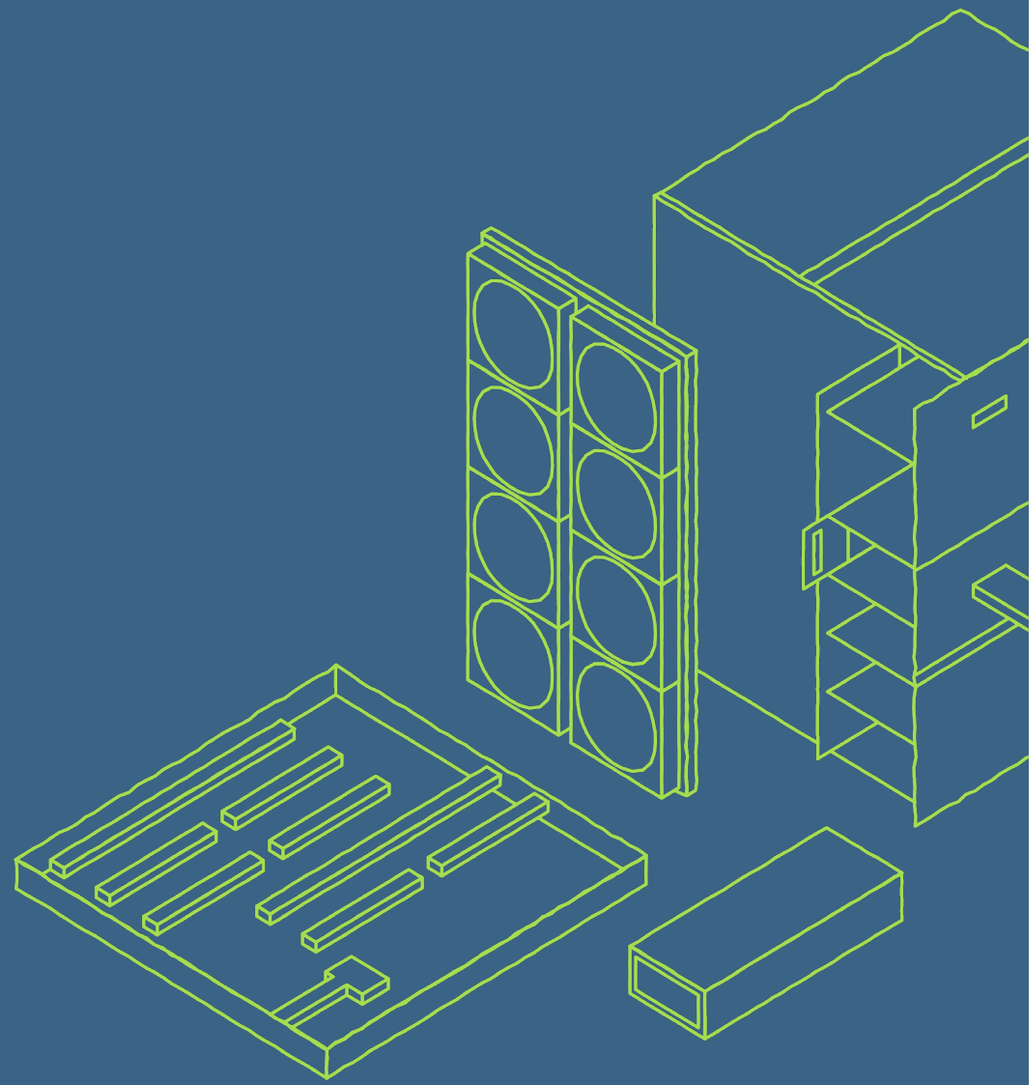
Status page

It's a good practice to publish a live report of your system so users can go to a particular URL, (a subdomain such as status.mycompany.com) to check the availability of services. Status.io is a good tool for this. If you want to provide greater transparency to your customers, you can include historic details about the uptime of the system.

A product support platform to integrate into your CI/CD pipeline:

Rookout
Rookout slashes debugging time and effort across the full product cycle from development through staging and production. Get data visibility and collection without restarts, redeployments, or added code.

Pitfalls to avoid



I'll reiterate here: don't ever lie. Once trust is altered, it's very difficult to recover from it.

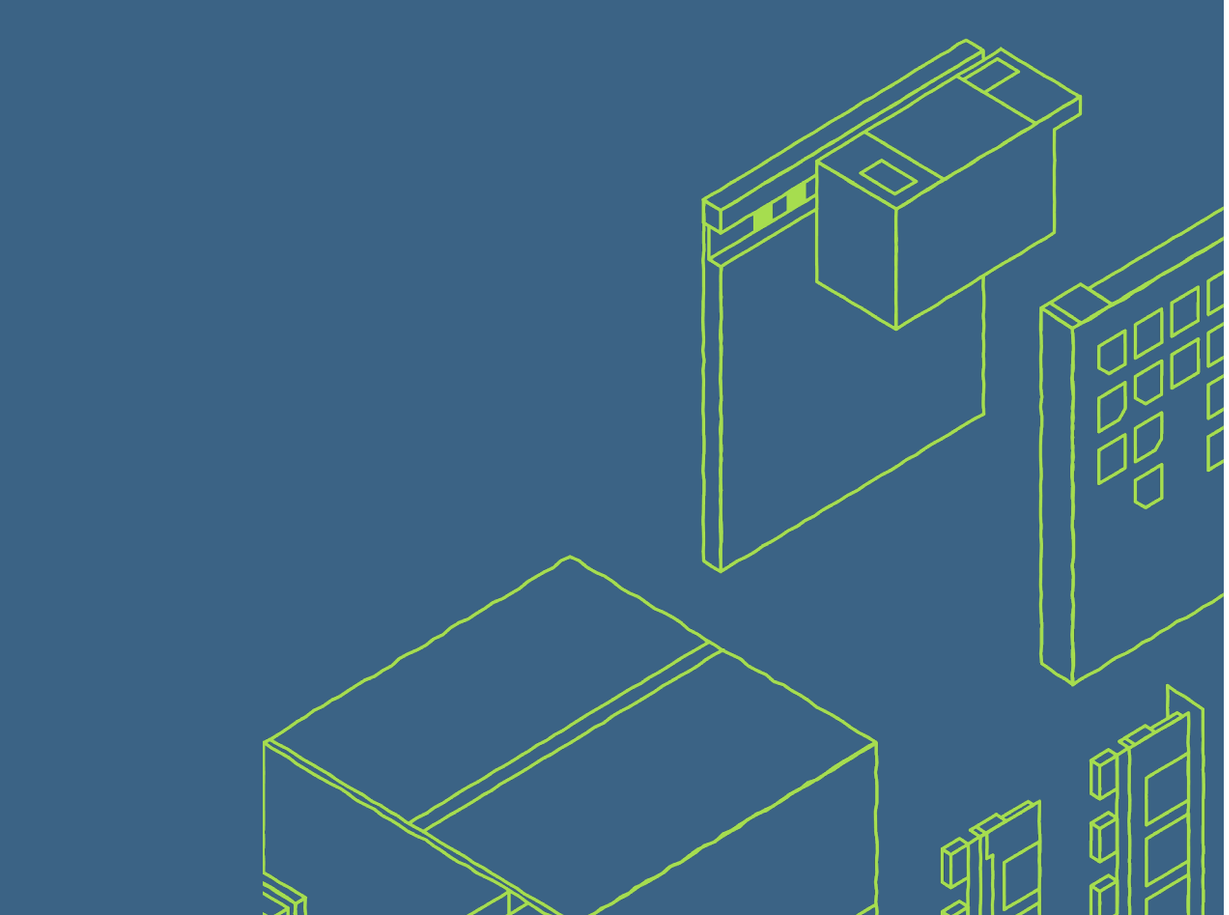
Don't bury the dirt behind evasive answers. Auditors have excellent skills when it comes to spotting these kinds of behaviors and will dig further down, potentially putting you in a delicate situation.

Be upfront about the things you know should be improved. No company has everything done perfectly. Everyone takes shortcuts to achieve a higher velocity. You need to be open about this but also know what needs to be improved and have a plan for it.

Be friendly and cooperative. A Tech DD is also a stressful process for the auditor, as the timeframe for execution can be quite short. Establishing a good human rapport with the auditor is always a good idea.

Be aware that auditors sometimes play mind tricks to get to the bottom of things. If faced with a defensive CTO, an auditor will most likely build a certain level of tactical empathy to gather more data. Some hard-to-detect techniques, such as mirroring inspired by the negotiation field, can be used to get to the level of information the auditor wants to acquire. The best way to deal with this is to be an open book and deliver the information the auditor needs, ideally before they even ask for it.

A strong foundation for your company



While Tech Due Diligence can be intimidating, this guide should reassure that you know what to cover and how to balance the different topics.

Regardless of an imminent Tech DD, you can use this guide as a reference to build strong foundations for your company and make sure you keep things in check.

With enough preparation and a good foundation you should be able to pass your next Tech DD without issue.

Author Information

EMAIL

johann@romefort.net

WEB

romefort.net

LINKEDIN

linkedin.com/in/romefort

TWITTER

@romefort