The secret to building high quality software at scale

How enabling developers to manage deployments as self-services allows you to build high quality software at scale faster.

codesphere

Executive Summary:

- Developers waiting on DevOps colleagues for deployments kills time2market
- Agile does not speed up development in rigid linear deployment environments
- Merging development and operations into DevOps teams only insources the bottleneck
- Kubernetes is hard and high maintenance
- New technologies enable developers to reclaim full ownership over their apps

Every company is a software company

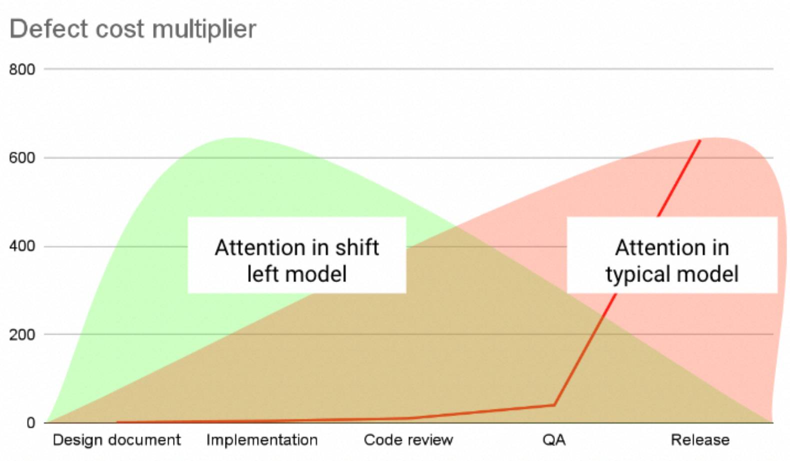
Software is eating the world. Building software in-house allows companies to create bespoke solutions that align precisely with their business processes and objectives. As digital (platform) revenues become more relevant throughout many industries, it is also important to capture that revenue in-house instead of handing off margins to external software vendors. Therefore, software development is a strategic imperative for enterprises seeking to optimize their operations and achieve long-term growth.

Despite the benefits there are common pitfalls for companies that are not natively technology companies:

- Development often takes longer than predicted. First, there's the planning and design phase, where teams must thoroughly analyze requirements, create detailed specifications, and architect the software.

- Next, testing and quality assurance processes are essential to identify and rectify bugs and ensure functionality and security.

- Additionally, integration with existing systems, compliance with industry standards, and user acceptance testing contribute to project timelines. Collaboration, communication, and coordination among team members also influence project duration.



- Finally, unexpected challenges and changes in requirements can further extend development time. Typically the later in a project's stage wrong or outdated requirements are noticed the more costly it becomes to fix them.

Agile & DevOps is not the answer

Agile development is widely recognized for addressing the intricacies and uncertainties of software development through its focus on iterative, flexible methods. This approach involves breaking projects into smaller, manageable parts that get released frequently, promoting ongoing feedback, and adjusting to evolving requirements. This allows teams to address issues early, streamline communication, and deliver value more efficiently.

One key characteristic of such an agile software development workflow is that it aims to release new features (or sub-features) often, actually the more **frequent** the better. In enterprise landscapes this is easier said than done. Software needs to pass through a variety of stages, that all require individual hardware to be **provisioned** and there are (rightfully so) gatekeepers for production systems. Typical software operations teams are inherently **not well equipped** for such a speed of iteration.

This leads to a scenario where developers frequently dedicate over one-third of their work hours initiating, managing, and waiting for deployments. In cases of incorrect deployments, they must rectify and repeat the process. All this proves frustrating for both individual developers and the organization as a whole. It results in reduced efficiency, longer wait times for clients, and unnecessarily elevated end-to-end software development costs.

DevOps was introduced as a way to avoid this costly division. Developers and operations join a single team and share the knowledge, allowing (some) developers to manage infrastructure provisioning without outside help. But let’s face it: Is that a feasible solution knowing how much domain specific know-how high performance computing managers and operations experts need on a daily basis?? Let’s dive deeper into this.

Obstacles to software provisioning as a self service

Provisioning is the process of creating and setting up IT infrastructure, and includes the steps required to manage user and system access to various resources. Provisioning is an early stage in the deployment of servers, applications, network components, storage, edge devices, and more.

And when thinking about provisioning one may have Kubernetes controlled containers in mind. So what is it and how does it work?

- **Traditional deployment era:** Early on, organizations ran applications on physical servers. There was no way to define resource boundaries for applications in a physical server, and this caused resource allocation issues. For example, if multiple applications run on a physical server, there can be instances where one application would take up most of the resources, and as a result, the other applications would underperform. A solution for this would be to run each application on a different physical server which is expensive.

- **Virtualized deployment era:** As a solution, virtualization was introduced. It allows you to run multiple Virtual Machines (VMs) on a single physical server's CPU. Virtualization allows applications to be isolated between VMs and provides a level of security as the information of one application cannot be freely accessed by another application.

Virtualization allows better utilization of resources in a physical server and allows better scalability because an application can be added or updated easily, reduces hardware costs, and much more.

The next level of virtualization is to manage it all in a container:

- **Container deployment era**: Containers are similar to VMs, but they have relaxed isolation properties to share the Operating System (OS) among the applications. Therefore, containers are considered more lightweight. Similar to a VM, a container has its own filesystem, share of CPU, memory, process space, and more. As they are decoupled from the underlying infrastructure, they are portable across clouds and OS distributions.

Containers have become popular because they provide extra benefits, including agile application creation and deployment, continuous development and integration, separation of concerns between Dev and Ops, observability, environmental consistency, portability, application-centric management, and efficient resource utilization. Unfortunately they have introduced a whole new world of skills and systems that many traditional developers have little or no experience in. Let’s take a look at what it takes to set up and maintain such systems.

Setting up containerized software landscapes is easy, with these points in mind

The provisioning process for cloud-based resources, including technologies like Docker and Kubernetes, involves several steps:

- **Resource Request:** The process typically begins when a development team or project requires additional resources, such as virtual machines or containers, to support their applications. This request is often made through a self-service portal or an infrastructure-as-code (IaC) script. In autoscaling cases applications themselves can trigger such resource requests when they experience high load.

- **Resource Allocation:** Once the request is made, the cloud provider allocates the requested resources. These resources may include virtual machines (VMs) for running applications or containers for packaging and deploying applications using technologies like Docker.

- **Containerization with Docker:** If Docker is part of the technology stack, developers package their applications and dependencies into Docker images. Docker allows for consistent and portable deployment, ensuring that the application runs reliably across different environments. Docker images contain a single application.

- **Orchestration with Kubernetes:** For containerized applications, Kubernetes (open-source Google project) is often used for orchestration. Kubernetes, being a cluster, multi container management solution that automates the deployment, scaling, and management of containerized applications. It ensures that all containers are running as expected, handles load balancing, and can scale the hardware running the application up or down based on demand.

- **Infrastructure as Code (IaC):** To manage Kubernetes build infrastructure, organizations commonly use IaC tools like Terraform or Ansible. These tools allow teams to define the desired infrastructure configuration in code, to provision and manage Kubernetes clusters and associated resources. These config files can be tens of thousands of lines of code.

- **Cluster Provisioning:** IaC scripts are used to provision the Kubernetes cluster, including master nodes and worker nodes. These scripts define the network configuration, security policies, and other parameters necessary for the cluster's operation.

- **Cluster Management:** Once the Kubernetes cluster is up and running, ongoing management tasks are essential. This includes monitoring cluster health, applying updates and patches, and scaling the cluster as needed to accommodate workload changes.

- **Application Deployment:** Developers deploy their containerized applications to the Kubernetes cluster using Kubernetes manifests or Helm charts. Kubernetes ensures that these applications are distributed across nodes, load-balanced, and can self-heal in case of failures. To work with Kubernetes locally during development additional software and high performance laptops are needed.

- **Monitoring and Logging:** Monitoring tools like Prometheus and Grafana are often used to collect metrics and visualize the health of the Kubernetes infrastructure and applications. Logging solutions like Elasticsearch and Kibana help track and analyze logs for debugging and troubleshooting.

- **Security and Access Control:** Kubernetes offers various security features, such as role-based access control (RBAC), network policies, and container runtime security. Managing these aspects is crucial to ensure the security of the infrastructure.

- **Scaling and Optimization:** Kubernetes allows for dynamic scaling of applications based on resource utilization. It's essential to monitor resource usage and optimize the cluster's capacity to minimize costs while ensuring performance. Scaling up horizontally (= across new / additional nodes) takes some time though.

- **Backup and Disaster Recovery:** Implementing backup and disaster recovery strategies for both applications and the Kubernetes infrastructure is vital to prevent data loss and minimize downtime.

By now the question is: How is that easy? Is that what a Developer is supposed to do if self-deployment shall become the norm? Doesn’t it require skills and knowledge and most of all time to manage all of that? So far the short answer is “yes” to all questions and that is why deployment on software defined infrastructures is typically a job of a DevOps or Ops specialist.

Additionally, it remains important to understand that even Kubernetes cannot do everything:

- **Kubernetes is not** a traditional, all-inclusive PaaS (Platform as a Service) system. Kubernetes provides the building blocks for building developer platforms, but preserves user choice and flexibility where it is important. It’s a cluster framework that takes months to configure properly.

- **Kubernetes does not** deploy source code and does not build your application. Continuous Integration, Delivery, and Deployment (CI/CD) workflows are determined by organization cultures and preferences as well as technical requirements; they are needed on top of Kubernetes.

That brings us to the heart of our point: How can you still use a containerized environment, while making it an easy to use self service for developers at the same time?
The answer is: Codesphere. Making containerized enterprise landscapes easy enough for anyone to set up and operate.

- **Why?** We want to empower developers to manage their own infrastructure needs. Codesphere aims to increase the efficiency and effectiveness of software development processes and provide agile teams with the necessary tools to bring innovations to the market faster and more effectively.

- **How?** By developing unique solutions for software development, deployment, and operation that are both efficient and user-friendly. We simplify software deployments to a point where no additional DevOps expertise is needed while maintaining full flexibility from Kubernetes.

- **What?** The Codesphere platform holistically supports the entire journey from code to cloud. Build from the ground up to enable developers to carry out the entire process as easy & efficient self-service.

High quality software at scale through developer self services

Enabling development to effectively self-provision their own infrastructure needs is the only way enterprise teams will be able to ship high quality software products at scale and reasonable prices.

Simplification is on the rise throughout the entire tech landscape and with the help of the right tools the upsides will be nothing short of game-changing. Looking back it will seem almost idiotic to have ever decoupled developers so far from the place where their code will ultimately run. Alongside some common best practice processes an infrastructure platform like Codesphere will speed up your team's output drastically.

Summary

1. A division between agile development and linearly working DevOps teams slows down significantly
2. Kubernetes is hard and high effort making it infeasible to turn entire development teams into DevOps experts
3. New tools like Codesphere help to overcome this challenge
4. Self-provisioning is the last hard hurdle to be solved to make development teams future ready