**What is provisioning in DevOps?**

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.

Provisioning and deployment: A lot of the time, people who work in or around IT use the two terms interchangeably. But they don’t mean the same thing.

At a high level, provisioning is the act of getting a device ready for a user. Deployment is the whole process of getting a device to a user.

Broadly speaking, provisioning entails defining the settings and software you want on the device, then implementing them (typically with an MDM solution). It is part of the deployment process; in most cases, you can’t do one without the other.

“So what?” you might ask. Why worry about fine differences in meaning when you’ve got work to do? We’d say that such distinctions are actually vital to getting that work done.

When you’re trying to explain something to an IT admin, manager, or executive—especially something as potentially complex as the deployment of Apple devices—you need to be sure that everyone involved in the project understands what you mean when you say “provisioning” versus “deployment.” Understanding that difference will help everyone communicate and think through the process.

Such clarity can also help you navigate change. Maybe you’ve previously been in charge of the provisioning process, but now you’re being asked to manage an entire deployment. Or maybe your team has grown, you no longer handle both, and you have to explain which parts of the process you want someone else to handle. Such changes require a common understanding of the terms you’re using.

So, yes, we think the difference in meaning between “provisioning” and “deployment” matters. Here’s how that difference can play out in the real world.

**What Is Provisioning?**

Modern provisioning starts with defining the desired settings and software necessary to meet the organization’s goals. Those calculations may require consulting with other teams outside of IT: the people team, the security team, procurement, networking, and—importantly—the teams that will actually use the devices you’re deploying.

There are three main approaches to provisioning:

**Fully pre-provisioned.** Devices are fully configured by IT; everything the user will need is preinstalled before they receive the device. After a device is delivered, MDM is mainly used for maintenance.

**Thin provisioning (a.k.a. middle-ground or hybrid).** IT pre-provisions only those settings and apps that are absolutely necessary from the moment the user receives the device. Users are responsible for things like creating user accounts; additional content may be provided via self-service.

**Zero-touch**. Devices are ordered and shipped directly from the vendor to the user, without any pre-provisioning. MDM is used to deliver apps and settings after the user opens the box, and that user plays a role in facilitating device configuration.

Fully provisioning devices requires the greatest time commitment from IT. The configuration steps must be determined and then repeated on every device. The other two models still require you to determine the provisioning steps, but reduce or remove the requirement for IT to configure each device directly.

Your choice among those three will be guided largely by the configuration needs you identified in that first consultative step and by what you want the experience to be like for users. (For more detail on this, see our post on managing large-scale deployments.)

**What is Continuous Monitoring in DevOps?**

Fundamentally, Continuous Monitoring (CM), sometimes called Continuous Control Monitoring (CCM), is an automated process by which DevOps personnel can observe and detect compliance issues and security threats during each phase of the DevOps pipeline. Outside DevOps, the process may be expanded to do the same for any segment of the IT infrastructure in question. It helps teams or organizations monitor, detect, study key relevant metrics, and find ways to resolve said issues in real time.

Continuous Monitoring comes in at the end of the DevOps pipeline. Once the software is released into production, Continuous Monitoring will notify dev and QA teams in the event of specific issues arising in the prod environment. It provides feedback on what is going wrong, which allows the relevant people to work on necessary fixes as soon as possible.

Continuous Monitoring basically assists IT organizations, DevOps teams in particular, with procuring real-time data from public and hybrid environments. This is especially helpful with implementing and fortifying various security measures – incident response, threat assessment, computers, and database forensics, and root cause analysis. It also helps provide general feedback on the overall health of the IT setup, including offsite networks and deployed software.

**Goals of Continuous Monitoring in DevOps**

* Enhance transparency and visibility of IT and network operations, especially those that can trigger a security breach, and resolve it with a well-timed alert system.
* Help monitor software operation, especially performance issues, identify the cause of the error and apply appropriate solutions before significant damage to uptime and revenue.
* Help track user behavior, especially right after an update to a particular site or app has been pushed to prod. This monitors if the update has a positive, negative, or neutral effect on user experience**.**

**Types of Continuous Monitoring**

**Infrastructure Monitoring:** Monitors and manages the IT infrastructure required to deliver products and services. This includes data centers, networks, hardware, software, servers, storage, and the like. Infrastructure Monitoring collates and examines data from the IT ecosystem to improve product performance as far as possible.

**Application Monitoring:** Monitors the performance of released software based on metrics like uptime, transaction time and volume, system responses, API responses, and general stability of the back-end and front-end.

**Network Monitoring:** Monitors and tracks network activity, including the status and functioning of firewalls, routers, switches, servers, Virtual Machines, etc. Network Monitoring detects possible and present issues and alerts the relevant personnel. Its primary goal is to prevent network downtime and crashes.

**Benefits of Continuous Monitoring**

* **Better Network Visibility and Transparency:** CM offers DevOps teams clarity on the state of the IT infrastructure by automatically collecting and analyzing data to reflect possible outages and important trends.
* **Facilitates Rapid Responses:** A primary aspect of CM is implementing an alert system that immediately notifies the right people the minute an IT incident emerges. This enables timely response to security threats or functional stop-gaps, minimizing damage and allowing faster restoration of the system to optimal operational levels.
* **Minimizes System Downtime:** Consistent system monitoring and quick, necessary alerts help maintain system uptime by raising the alarm when there is a service outage or any application performance issues.
* **Assists with Healthy Business Performance:** Reduction in system downtime also minimizes negative impact on customer experience, thus safeguarding the organization against losses in revenue or credibility. As mentioned before, Continuous Monitoring tools can also be used to track user reactions to software updates, which is useful for several teams – development, QA, sales, marketing, customer service, etc.

**Risk Management and Continuous Monitoring**

There are numerous tools for every stage of Continuous Monitoring in DevOps. However, before selecting tools, organizations, and DevOps teams must conduct adequate risk assessment and formulate a risk management plan. Developers can only implement an appropriate CM system after a thorough evaluation of compliance systems, governance, and risk factors. These tend to be quite different between organizations depending on their nature; e.g., a private company will have a different view of risk than a government organization.

To facilitate understanding of these metrics, consider asking the following question when looking for tools to implement CM:

* What is the extent of risk that the organization can withstand and recover from?
* What are the parameters by which to calculate risk?
* For each parameter, is it possible to assign values that denote the highest potential risk?
* What is the level of confidentiality required by the data collected and generated by the organization?
* What are the consequences of security breaches, hardware, or software failure?

**Best Practices for Continuous Monitoring in DevOps**

**Decide what to monitor:** Different organizations need to monitor different aspects of their IT landscape. Primarily, the targets are categorized into the following:

1. Server status and health

2. Application performance log

3. System vulnerabilities

4. Development milestones

5. User activity/behavior

As far as possible, try to track parameters belonging to each category.

Use a Full Stack Monitoring Tool: The chosen Configuration Management tool should monitor the entire IT stack end-to-end to provide alerts and resources for issue resolution. In terms of what they should cover, here is a cursory list:

**Infrastructure Monitoring:** Tool must monitor:

Server Availability

CPU & Disk Usage

Server & System Uptime

Response Time to Errors

Storage

Database Health

Storage

Security

User permissions

Network switches

Process level usage

Relevant performance trends

**Network Monitoring:** Tool must monitor:

Latency

Multiple port level metrics

Server bandwidth

CPU use of hosts

Network packets flow

Application Monitoring: Tool must monitor:

availability

error rate

throughput

user response time

pages with low load speed

third-party resource speed

browser speed

end-user transactions

SLA status