Unit 5: Deployment of Enterprise applications:

Key requirements in deployment - Stability, capacity, Security.

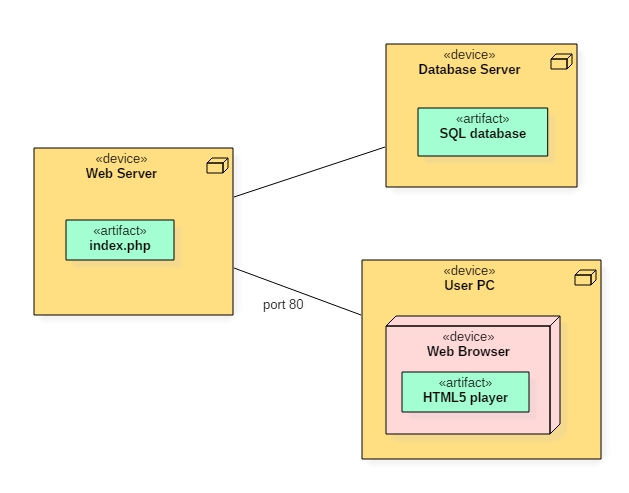
Availability, Network,

Availability, and

Transparency

What is an enterprise deployment?

The term Enterprise Software Deployment refers to **all of the activities that must take place to make a software application available to end users**. Enterprise Software Deployment technically begins after a software application has been written, tested, packaged and delivered to an enterprise



What is deployment of an application?

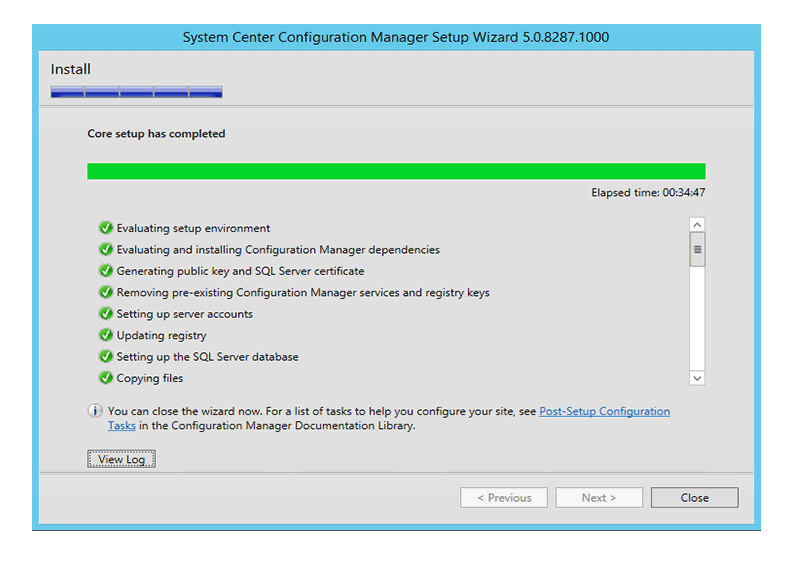
Application Deployment, also known as Software Deployment, is **the process of installing, configuring, updating, and enabling one application or suite of applications that make a software system available for use**, like facilitating a certain URL on a server.

Example of software deployment could be **when a user downloads a mobile application from the Integration Store and installs it onto their mobile device**. To summarize, a software release is a specific version of a code and its dependencies that are made available for deployment.

**Top 5 Software Deployment Tools for Enterprise**

[**1. SCCM**](https://docs.microsoft.com/en-us/sccm/)

Microsoft's System Center Configuration Manager is a tool for distributing software, patches, remote control, OS deployments, etc. It works for Windows based PC and also from iOS and Android devices.



**Pros:**

* Comprehensive management over multiple platforms

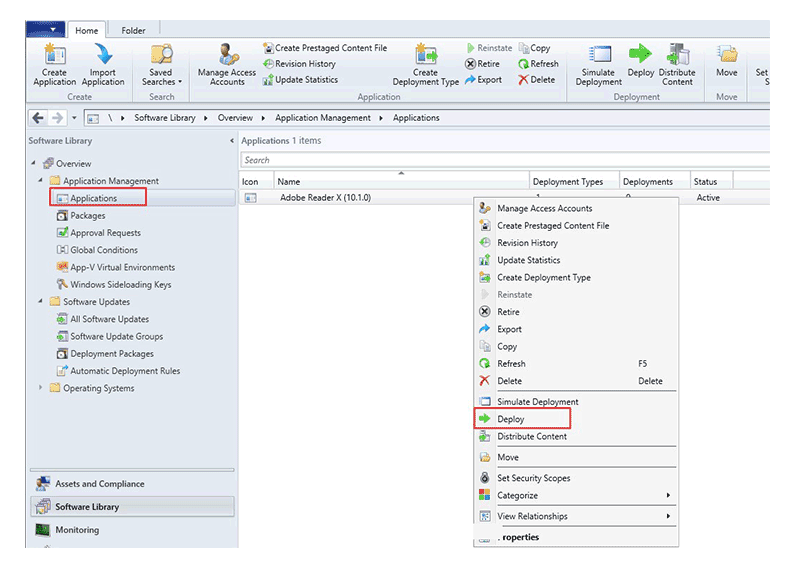
**Cons:**

* No warning provided for failed deployments or installations.

Below are the steps to deploy Software with SCCM:

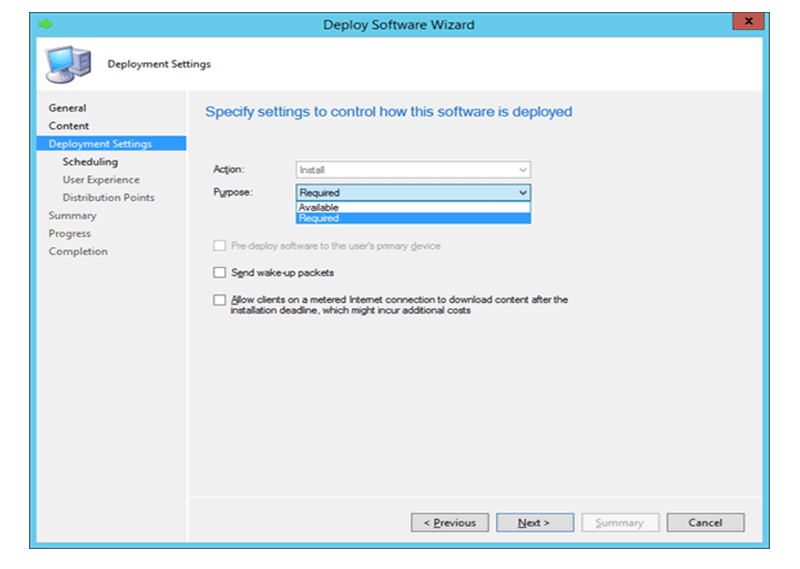
**Step 1: Choose the Application to Deploy**

An Adobe application is used as an example in the tutorial. You have to set up SCCM as per your needs. Then access Applications and select the software Adobe. Then press Deploy.



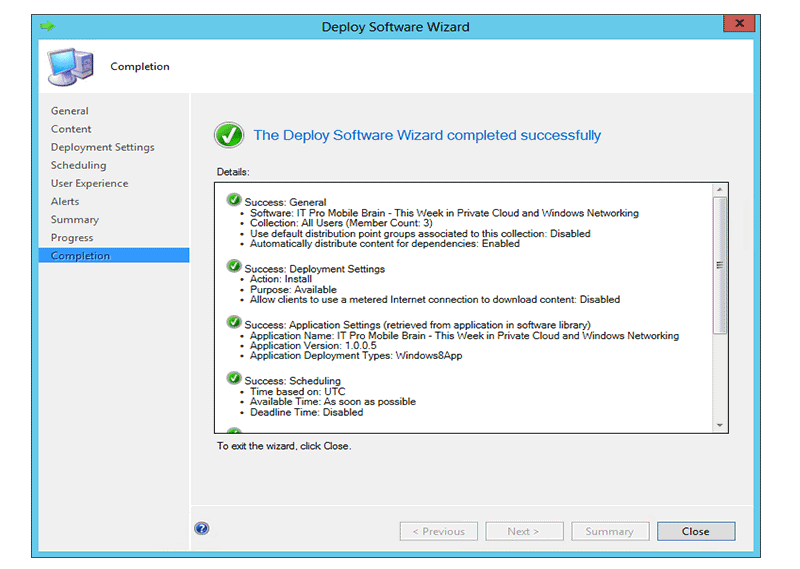
**Step 2: Specific Settings for Deployment**

The Deploy Software Wizard has to be configured now. We have named our target distribution group as BPO users. Hit Next to continue.



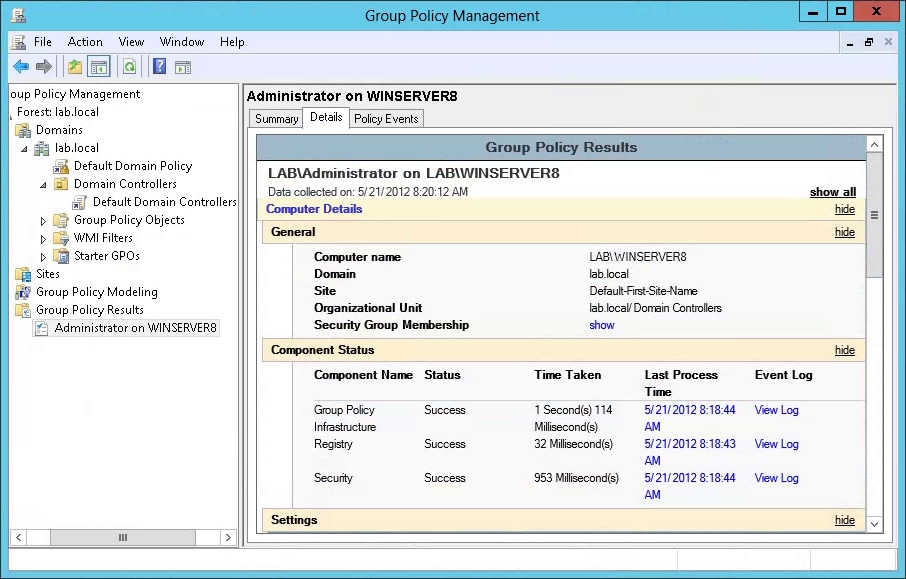
**Step 3: Deploy Software Successfully**

The Deployment Settings need to be defined depending on the actions you want to perform on the user end. Here we are installing the software by choosing Install. You can also set the option of user approvals for installations. Select "Next" and the software is deployed to the chosen target.



[**2. Group Policy**](https://technet.microsoft.com/en-us/library/cc753298(v=ws.11).aspx)

Windows server can utilize an extension called Group Policy to remotely control and deploy software and configurations to a user group. It uses Group Policy Objects to configure and control user devices.



**Pros:**

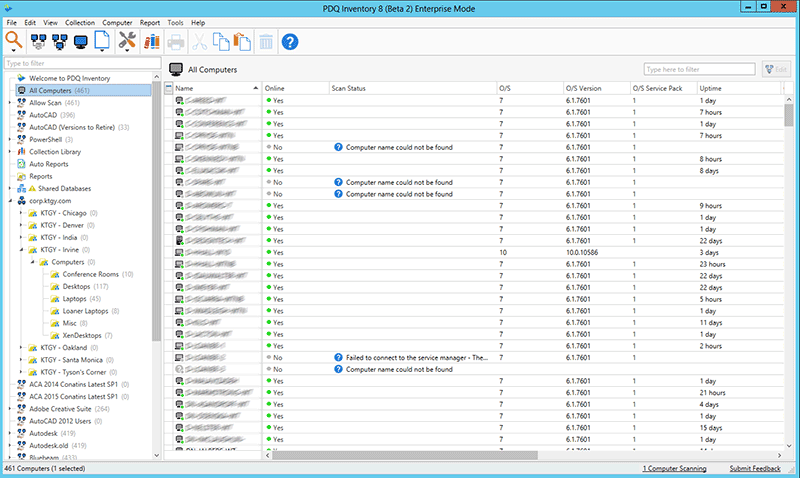
* Secured way of deploying over a network

**Cons:**

* Organizing and handling large number of GPOs is difficult

[**3. PDQ Deploy**](https://www.pdq.com/pdq-deploy/)

Enterprises can use a central computer to remotely deploy MSI, EXE, MSP batch installations through PDQ Deploy.



**Pros:**

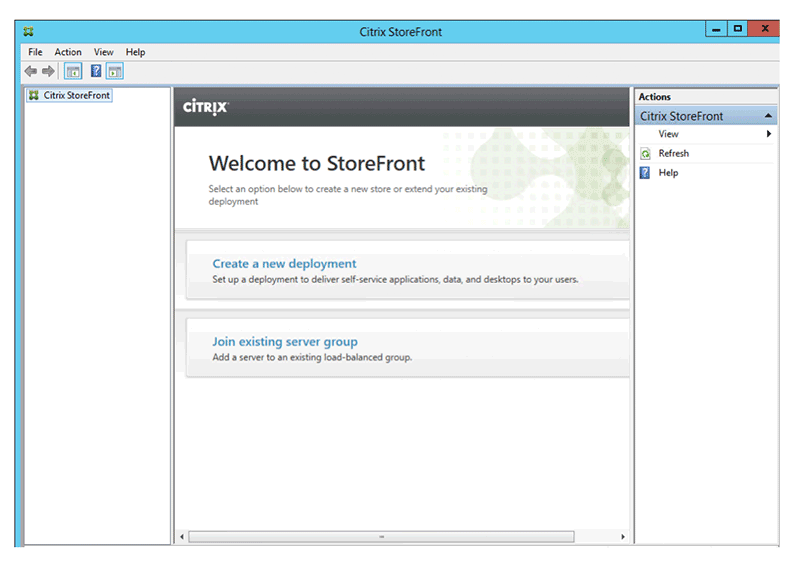
* Applications features can be edited before deploying

**Cons:**

* Features like automated Windows update repository are missing

[**4. Citrix**](https://www.citrix.com/)

Citrix servers provide virtual desktop sessions, software deployments, SaaS applications distributions and more to multiple types of devices.



**Pros:**

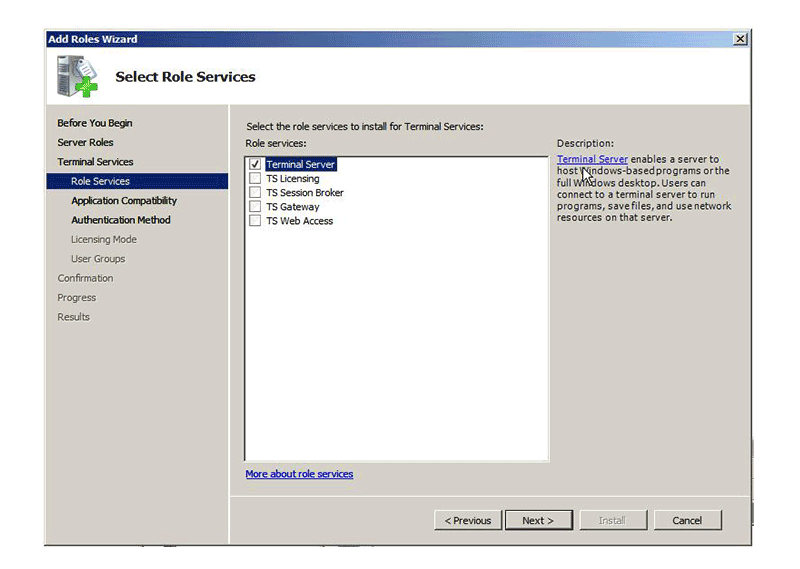
* Suitable for apps and mobile applications

**Cons:**

* It is a costly solution

[**5. Windows Terminal Server**](https://technet.microsoft.com/en-us/library/cc755399(v=ws.10).aspx)

Windows Terminal Server or Remote Desktop Services can deploy and control user end devices through a number of ways. Applications first need to be installed on server for distribution.



**Pros:**

* Does not require high-end hardware

**Cons:**

* Not compatible with all kinds of software

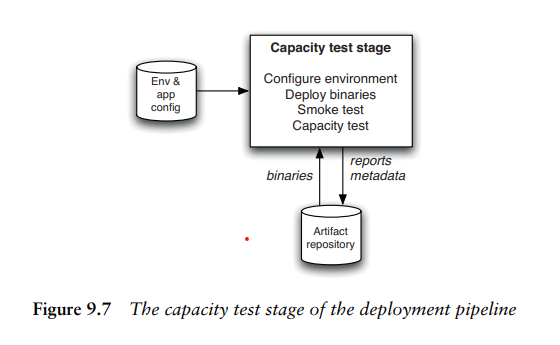
Key requirements in deployment:

**Stability and Capacity**

Most applications need to meet some minimum capacity threshold. Most modern commercial applications will be servicing many concurrent users, and will therefore be required to scale to meet their peak demand profile while delivering acceptable performance. During development, what we need is the ability to assert that our application will achieve the capacity required by the customer. While capacity-related non-functional requirements are an important facet of the development of a project, it is important to specify what “good enough” means in some quantifiable measure. These measures should be evaluated by automated tests of some kind that are run as part of the deployment pipeline. That means that every change that passes the commit tests and acceptance tests should have automated capacity tests run against it. Thus it becomes possible to identify the moment of introducing a change that significantly affects the application’s capacity. Passing the automated capacity tests, with the sweet spot clearly delineated by their success criteria, ensures that the capacity requirements are met. In this way, we guard against over-engineered solutions to the capacity problem. We always apply the dictum that we will do the minimum amount of work to achieve the result we are aiming for, as implied by the YAGNI (“You Ain’t ’Gonna Need It”) principle. YAGNI reminds us that any behavior we add defensively is potentially wasted effort. Applying Knuth’s dictum, optimizations should be deferred to the point when it is clear that they are required, deferred until the last responsible moment, and targeted based on runtime application profiling so as to attack bottlenecks in descending order of importance. As ever, our goal with any testing is to fail as quickly as possible after a change breaking our assumptions is introduced. In this way, the change is easily identified and quickly fixed. However, capacity tests are often relatively complex and can take a long time to run. If you are lucky enough to be able to prove that your application meets its performance goals within a few seconds, add your capacity tests to the commit testing stage so you can get immediate feedback on any problems. However, in this case beware of any technology that relies on runtime optimizing compilers. The runtime optimizations in .NET and Java take many iterations to stabilize, and sensible results can only be gathered after several minutes of “warm-up.” A similar strategy can be useful for protecting known performance hot spots from getting worse over time as the code develops. When such a hot spot is identified, create a “guard test” that runs very quickly as part of you commit test cycle. Such tests act as a kind of performance smoke test—they aren’t going to tell you that your application meets all of its performance criteria, but they may highlight trends in the wrong direction and let you tackle them before they become a problem. However, watch out that you don’t introduce untrustworthy tests that fail intermittently with this strategy. Most capacity tests, though, aren’t candidates for the commit stage of your deployment pipeline. They usually take too long and require too many resources to run. Adding capacity tests to the acceptance stage is feasible if the capacity tests remain fairly simple and don’t take too long to run. On the whole, though, we don’t recommend adding capacity tests to the acceptance test stage of your deployment pipeline. There are several reasons:

• To be really effective, capacity tests need to be run in their own special environment. Trying to figure out why the latest release candidate failed its capacity requirements so badly can be quite costly if the real reason was that some other automated tests were running simultaneously on the same environment. Some CI systems allow you to specify target environments for tests. You can use this feature to partition capacity tests and run them in parallel with acceptance tests.

• Some types of capacity test can take a very long time to run, resulting in an untenable delay before getting an acceptance test result. Many activities downstream from acceptance testing can be done in parallel with capacity testing, such as demonstrating the latest working software, manual testing, integration testing, and so forth. Gating these on a successful capacity test run is unnecessary and, for many projects, inefficient.

• For some projects, it does not make sense to run capacity tests as frequently as acceptance tests. In general, apart from the performance smoke tests we have described, we prefer to add automated capacity testing as a wholly separate stage in our deployment pipeline. How this capacity stage of the pipeline is treated differs somewhat from project to project. For some projects, it makes sense to treat it in a way similar to the acceptance test stage—as a fully automated deployment gate. That is, unless the tests in the capacity test stage all pass, you can’t deploy the application without a manual override. This is most appropriate for high-performance or large-scale applications that are simply not fit for purpose if they do not meet a well-understood threshold of capacity. This is the most rigorous model for capacity testing that, on the face of it, seems optimal to most projects. However, this is not always the case. If there are real issues of throughput or latency, or information that is only relevant or accurate for specific windows of time, automated tests can act very effectively as executable specifications that can assert that the requirement is met. At a high level, the acceptance test stage in the deployment pipeline is a template for all subsequent testing stages, including capacity testing, as shown in Figure 9.7. For capacity tests, as for others, the stage begins by preparing for deployment, deploying, then verifying that the environment and application are correctly configured and deployed. Only then are the capacity tests run. 

**Security**

To deploy secure applications, you must have a well-defined software development lifecycle, with appropriate security checks during the design, development, testing, and deployment stages. A layered system architecture that uses standardized frameworks for identity, authorization, and access control to be used.

**Automate secure releases**

Without automated tools, it can be hard to deploy, update, and patch complex application environments to meet consistent security requirements. Therefore, build a CI/CD pipeline for these tasks, which can solve many of these issues. Automated pipelines remove manual errors, provide standardized development feedback loops, and enable fast product iterations. For example, Cloud Build [private pools](https://cloud.google.com/build/docs/private-pools/private-pools-overview) let you deploy a highly secure, managed CI/CD pipeline for highly regulated industries, including finance and healthcare.

Automation to scan for security vulnerabilities when artifacts are created. You can also define policies for different environments (development, test, production, and so on) so that only verified artifacts are deployed.

**Ensure that application deployments follow approved processes**

If an attacker compromises your CI/CD pipeline, your entire stack can be affected. To help secure the pipeline, you should enforce an established approval process before you deploy the code into production.

If you plan to use Google Kubernetes Engine (GKE) or Anthos, you can establish these checks and balances by using [Binary Authorization](https://cloud.google.com/binary-authorization/docs/overview). Binary Authorization attaches configurable signatures to container images. These signatures (also called *attestations*) help to validate the image. At deployment, Binary Authorization uses these attestations to determine that a process was completed earlier. For example, you can use Binary Authorization to do the following:

* Verify that a specific build system or continuous integration (CI) pipeline created a container image.
* Validate that a container image is compliant with a vulnerability signing policy.
* Verify that a container image passes criteria for promotion to the next deployment environment, such as from development to QA.

**Scan for known vulnerabilities before deployment**

An automated tool that can continuously perform vulnerability scans on container images before the containers are deployed to production.

Use [Container Analysis](https://cloud.google.com/container-analysis/docs) to automatically scan for vulnerabilities for containers that are stored in [Artifact Registry](https://cloud.google.com/artifact-registry) and [Container Registry](https://cloud.google.com/container-registry). This process includes two tasks: scanning and continuous analysis.

To start, Container Analysis scans new images when they're uploaded to Artifact Registry or Container Registry. The scan extracts information about the system packages in the container.

Container Analysis then looks for vulnerabilities when you upload the image. After the initial scan, Container Analysis continuously monitors the metadata of scanned images in Artifact Registry and Container Registry for new vulnerabilities. When Container Analysis receives new and updated vulnerability information from [vulnerability sources](https://cloud.google.com/container-analysis/docs/vulnerability-scanning#sources), it does the following:

* Updates the metadata of the scanned images to keep them up to date.
* Creates new vulnerability occurrences for new notes.
* Deletes vulnerability occurrences that are no longer valid.

**Monitor your application code for known vulnerabilities**

It's a best practice to use automated tools that can constantly monitor your application code for known vulnerabilities such as the [OWASP Top 10](https://owasp.org/www-project-top-ten/). For a description of Google Cloud products and features that support OWASP Top 10 mitigation techniques, see [OWASP Top 10 mitigation options on Google Cloud](https://cloud.google.com/architecture/owasp-top-ten-mitigation).

Use [Web Security Scanner](https://cloud.google.com/security-command-center/docs/concepts-web-security-scanner-overview) to help identify security vulnerabilities in your App Engine, Compute Engine, and Google Kubernetes Engine web applications. The scanner crawls your application, following all links within the scope of your starting URLs, and attempts to exercise as many user inputs and event handlers as possible. It can automatically scan for and detect common vulnerabilities, including cross-site scripting (XSS), Flash injection, mixed content (HTTP in HTTPS), and outdated or insecure libraries. Web Security Scanner gives you early identification of these types of vulnerabilities with low false positive rates.

**Control movement of data across perimeters**

To control the movement of data across a perimeter, you can configure security perimeters around the resources of your Google-managed services. Use VPC Service Controls to place all components and services in your CI/CD pipeline (for example, Container Registry, Artifact Registry, Container Analysis, and Binary Authorization) inside a security perimeter.

VPC Service Controls improves your ability to mitigate the risk of unauthorized copying or transfer of data (data exfiltration) from Google-managed services. With VPC Service Controls, you configure security perimeters around the resources of your Google-managed services to control the movement of data across the perimeter boundary. When a service perimeter is enforced, requests that violate the perimeter policy are denied, such as requests that are made to protected services from outside a perimeter. When a service is protected by an enforced perimeter, VPC Service Controls ensures the following:

* A service can't transmit data out of the perimeter. Protected services function as normal inside the perimeter, but can't send resources and data out of the perimeter. This restriction helps prevent malicious insiders who might have access to projects in the perimeter from exfiltrating data.
* Requests that come from outside the perimeter to the protected service are honored only if the requests meet the criteria of access levels that are assigned to the perimeter.
* A service can be made accessible to projects in *other* perimeters using [perimeter bridges](https://cloud.google.com/vpc-service-controls/docs/share-across-perimeters).

**Encrypt your container images**

In Google Cloud, you can encrypt your container images using [customer-managed encryption keys (CMEK)](https://cloud.google.com/kms/docs/cmek). CMEK keys are managed in Cloud Key Management Service (Cloud KMS). When you use CMEK, you can temporarily or permanently disable access to an encrypted container image by disabling or destroying the key.

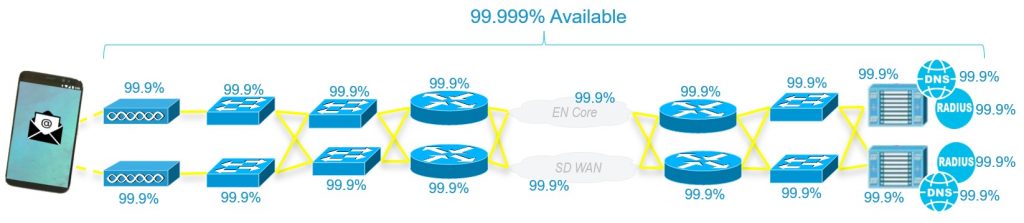
**Network Availability**

The best-known metric of network availability is known as “five [nines](https://en.wikipedia.org/wiki/High_availability#%22Nines%22)”. What five nines means is that the end-user perceives that their application is available 99.999% of the time. This permits only 5.26 minutes of downtime a year. Depending on the application and network topology, this can be a very stringent standard.

Consider Figure 1 below which shows serially connected routers, switches, access points, servers, and transited clouds.  When these ten elements are connected without any redundancy, each of these elements must be up and available 99.9999% (or six nines) of the time for the end-user to perceive five nines of availability.  As six nines allows only 32 seconds of downtime, having a single reboot a year could prove problematic.

Figure 1: Serial Transport Availability

The good news is that with the proper network, application, and services architecture, the individual devices making up the Internet do not need to support six nines of availability. All we need to do is add some redundancy. The following network design includes such a well-architected redundancy-based design. For this network design, if each element is fully independent, and if each element is available just 99.9% of the time, then the end-user will experience 99.999% availability.

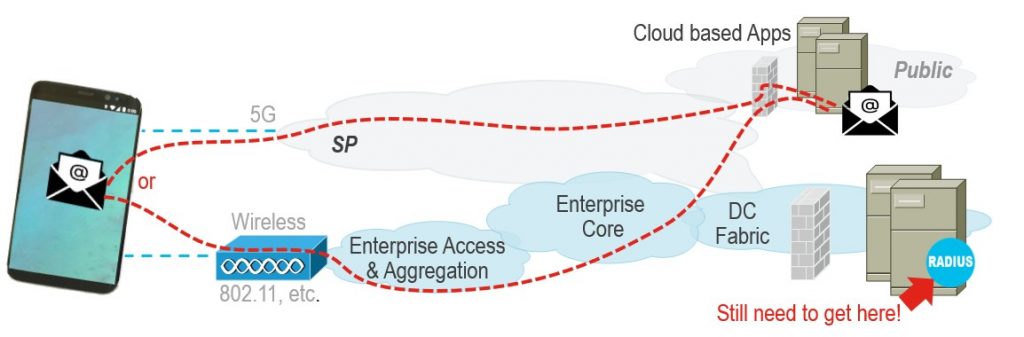
Figure 2: Parallel Transport Availability

Despite the user’s experience being identical, the difference between the two figures above is huge. We have reduced the availability requirements of all component parts by three orders of magnitude. And we have made something highly reliable from less reliable parts. This really shouldn’t be surprising however. From its very beginnings, the Internet was [designed to be available](https://www.rand.org/about/history/baran.html) even when devices were lost to nuclear attacks.

In the decades since the Internet’s conception, Cisco has [documented](https://www.cisco.com/web/IT/unified_channels/area_partner/cisco_powered_network/net_availability.pdf) many technologies and approaches to achieving a very high degree of availability. A small subset of these includes quickly converging routing and switching protocols, device and link redundancy, and boot time reduction. But such technologies and approaches are only part of the availability equation. Network operators have the ultimate say in deploying these technologies to maximize network availability. Strategies include the distribution of application servers across geographically and organizationally diverse datacenters, as well as redundancy of access and core networks all the way to ensuring that fiber-optic cables from different service providers don’t run in the same fiber conduit. These strategies are proven to be effective at providing high availability.

The result of all this good network design and planning is that the [majority](https://www.prescientsolutions.com/blog/prevent-7-common-causes-network-failures/) of application availability failures don’t come from equipment failures. Instead they come from equipment misconfiguration. Protecting the consistency of the network configuration is non-trivial and becomes more difficult as you add new technologies to the network. In fact, protecting network consistency is a key reason network operators are choosing to deploy controllers to manage device configuration based on higher level expressions of [intent](https://www.cisco.com/c/en/us/solutions/intent-based-networking.html). One of the main goals of network controllers is to automatically ensure correct and consistent configuration of all of the equipment in the network.

Intent, while very useful in this role, might not address every dimension of application availability. Consider the picture below of an Enterprise network integrated with a Public-Cloud topology.

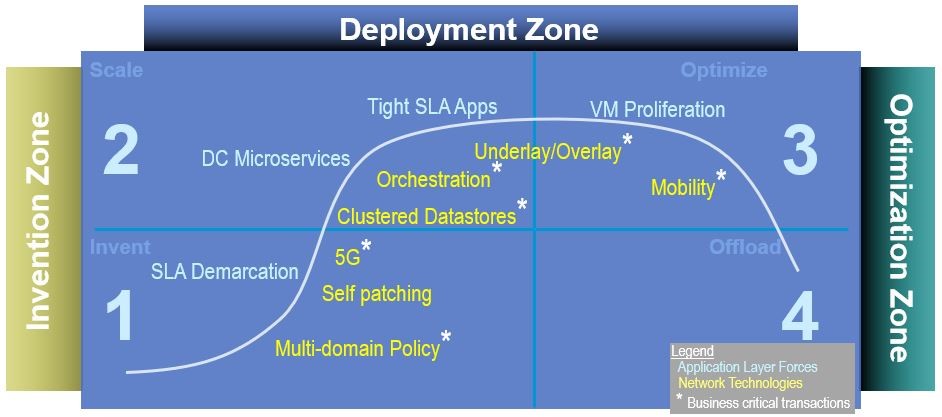
Figure 3: Public Cloud Apps need Enterprise Authentication

In this network design, the Public cloud-based applications accessed solely through cellular data do not just depend on the cloud. They still depend on the accessibility of an Enterprise’s RADIUS Authentication infrastructure. In other words, at best a cloud-based application will only be as available as access to your Enterprise Data Center. This is a nuance which very few end-users will be able to recognize or troubleshoot as a cause of availability issues.

## ****New Technologies Add Risks to Availability****

It is not just the Enterprise’s Authentication infrastructure which we need to consider when thinking about the future of availability. There is a set of forces which are changing network design. Geoffrey Moore has done much work describing the continuous technology [invention and deployment cycle](http://www.kilkku.com/blog/2014/07/moore%E2%80%99s-category-maturity-life-cycle-and-innovation-types/). Based on this, it is best to think of the network as a continually changing entity.

Figure 4 below shows a subset of the forces and technologies which are top-of-mind Enterprise network design. Each of these have the opportunity to improve or degrade application availability if they are not taken into consideration during the network design.

Figure 4: Emerging Technologies Use Controllers

With the advent of Software-Defined Networking ([SDN](https://searchnetworking.techtarget.com/definition/SDN-controller-software-defined-networking-controller)), the emergence and growth of new types of controllers is a trend which broadly impacts network availability calculations. Above in Figure 4, you can see a number of starred\* technologies. Each star represents a new controller involved in the establishment and maintenance of an application flow. And the result of each star is the addition of a transactional subsystem which impacts the calculation of network availability.

What are examples of these transactional subsystems? Historically we have depended on transactional subsystems as DNS, BGP, DHCP, and Wireless LAN Controllers. As networks evolve, we are seeing the proposal or introduction of new transactional subsystems such as [OpenFlow](https://www.researchgate.net/publication/301410178_Controller_failover_for_SDN_enterprise_networks) servers. We are also seeing the evolution of existing transactional subsystems such as RADIUS/Identity. The RADIUS/Identity evolution is quite important here. The evolution of user and workload identification is becoming more complex as cloud systems are integrated into the Enterprise. It is worth considering the impacts to application availability as corporate access control gets more deeply integrated into the cloud via technologies like [Azure AD](https://azure.microsoft.com/en-us/services/active-directory/), [Google IAP](https://cloud.google.com/iap), [SPIFFE](https://spiffe.io/), and [ADFS](https://www.okta.com/blog/2018/06/what-is-adfs/).

**Transparency**

Every month-and-a-half, Mozilla releases a new version of their Firefox web browser. Internally, the company establishes an R&D [commitment to delivery](https://www.panaya.com/sap/), and externally it publishes the dates and content of future releases. Subsequently, the pressure to meet goals is high and demands a lot of attention and accuracy in the planning stage. During the release timeline, however, things change and the organization needs to respond. Whether it is a competitor releasing a new feature or a technology that can improve performance, these changes affect the release and, if not administered correctly, can create disorder and confusion.

Teams that want to [achieve agility](https://www.panaya.com/blog/testing/accelerate-application-delivery-with-agile-alm/), and as a result benefit from being able to introduce new features in the middle of a release cycle, need to continually know and understand their status—delivery and quality-wise. Only a crystal-clear view allows teams and management to manage risks while deciding whether or not to push a new capability. Being able to reflect the state of the release at any point in time is a game changer for R&D and delivery teams, as well as leaders who want to take their dynamism to the next level.

Transparency: A Key Requirement to Successful Delivery

R&D teams planning their release take into account many parameters—capacity, requirements research, availability of third parties (parallel teams), and more. When the planning phase is over, team members should each have a clear understanding of the features they are working on and delivery deadlines. From then on, until the feature is completed, the information about the progress of development and testing usually remains clear only to developers, feature owners, and QA engineers who are part of the team, and unclear to any other interested parties.

How Transparency Can Transform your Work Environment

Without transparency, release managers who want to understand the convergence of the release must approach the team leader/feature owner and ask for visibility. Project managers who want to check on the status of new features need to inquire about this with QA engineers. In order to report on release quality, QA managers need to collect data from all involved teams all along the lifecycle, analyze and compile it so they can produce a clear view on the status.

Consider the case of HP Application Lifecycle Management (HP ALM), built over the last 20 years. There were several issues with its 11th version, identified as high risk only a few days prior to the release. The QA manager worked with four feature teams on the release. However, due to lack of transparency, he was not fully aware of the delivery details along the way. The issues were actually in the system for a while, but were never reported. Nonetheless, they were serious enough for HP to release a critical patch with fixes less than two weeks after the original release of the version. This shows the importance of teams consistently reflecting the status of an application in a shared location—physically or technologically— so stakeholders can access, track, and extract the information they need at the time they need it.

Transparency done well produces a trustful environment and simpler collaboration amongst the team and with others responsible for the development and release process. More importantly, transparency creates confidence for making informed decisions about changes in scope or prioritizing throughout the release. This confidence is transmitted over to organizational leaders and upper management who then give freedom of action to R&D, [leading to a healthier work environment, better products, and higher quality](https://www.panaya.com/blog/modern-alm/agile-in-the-enterprise/).

Transparency and Visibility: Not More of the Same

When we say “transparency,” don’t think only “visibility.” Visibility isn’t enough. [Research](https://researchspace.auckland.ac.nz/handle/2292/22092) done about transparency in software development ranks understandability—the ability to comprehend the information displayed—as the second most common problem for project reflection. Teams that share information should understand why consumers require the info and build the dashboard or report accordingly.

Sharing partial information is another concern when it comes to transparency. In 2015, a month before a well-known company in the Bay Area finalized their acquisition by an enterprise-sized company, their R&D team announced the release of a new mobile application with improved features and user experience. Management began bragging about the app in board meetings with their new owners, but were disappointed to discover after that only an iOS application would be released. The new Android app release was planned for a few months later. The board of the enterprise decided to revoke the acquisition claiming the management team had been irresponsible in not knowing the specifics of the application release.

Sharing partial information defeats the purpose of eliminating the element of surprise, which is one of the main goals of transparency. Furthermore, such behavior creates distrust, at the very least, and at worst, severely impacts business operations.

Transparency: Integrating it into the Continuous Delivery Process

A tool as simple as a **public scrum board** creates openness and insight into the team’s work and process. As a result, the team and external individuals may see, at a glance, whether they are on track to meet their sprint goals, have too much work in progress, or are blocked on one or more stories. Based on the status on the board, teams can decide where to focus their effort and change plans on the fly. Using the same board, the change manager gains control of the features developed and is able to keep track on their state and progress.

Another useful tool for sharing information is a periodic**delivery and quality status report**.

Organizations now understand that while awesome new features might bring new customers, quality is the most important facet for preserving them. The focus on the product’s quality is higher than ever and has a lot of attention. Teams that are confident in their deliverables’ quality and share their confidence among the whole R&D group, have better ability to accept or reject change requests coming from the product manager or from customers.

**Dashboards and graphs** of quantitative information facilitate an accurate discussion about the release state. Release and change managers require precision in order to plan and change content during the release and these views enable this capability. But dashboards have a tricky element—relevancy.

Dashboards must display respective and practical information which stakeholders can utilize and benefit from so it is important to maintain different views for different stakeholders, so the information is relevant and clear.

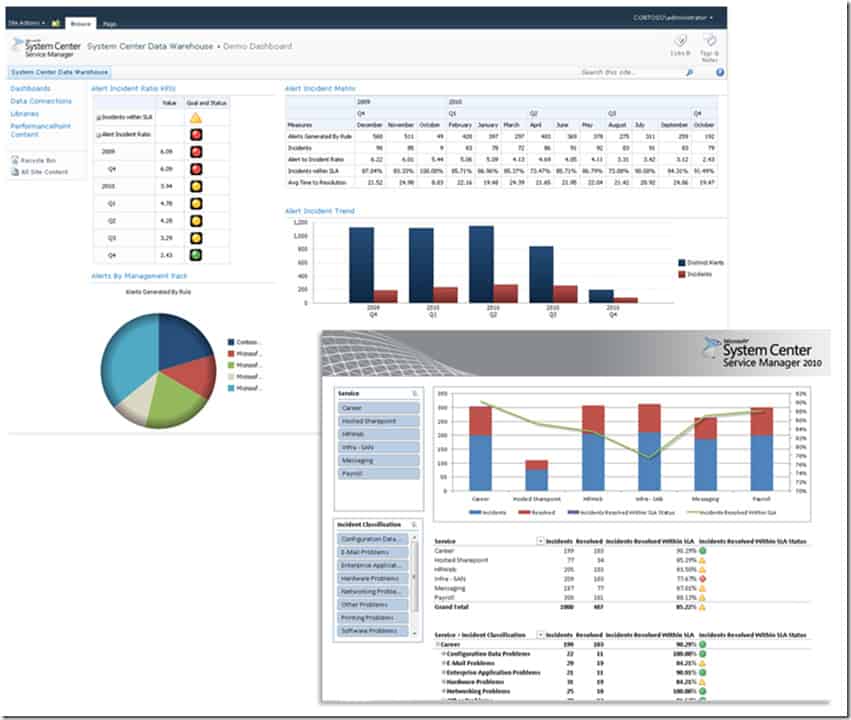
In the same research on transparency mentioned above, another significant deficiency is sharing the right information to the wrong people. If managers or parallel teams can’t learn the information from the view, they either approach team members to acquire this information—which defeats the purpose of the view—or worse, ignore the information displayed.

When developing a feature that is dependent on services from another team, transparency and clarity become critical. Teams cannot commit to deliver functionality if they have no visibility of its infrastructure status. Communication and collaboration culture can only be improved when the status of both teams is completely transparent and the teams understand and trust the others’ objectives and processes.

**Transparency: Sharing Across All Stakeholders**

As said, it is important to understand the players who require the shared information, what they require it for, and how they make use of it, in order to build the right view for the right individual. These include release managers, project managers, upper management, and even end-users, all of whom will benefit from transparency.

The top consideration of **release managers** is whether to approve a release or delay it. Convergence, quality, risks, and service level are the release metrics they plan and measure to monitor its status. Understanding this need allows R&D teams to create a view that’s beneficial to the release manager and make them feel part of the team. Release managers in many organizations have a dashboard designed specifically for them combining views on all the parameters affecting the release and more aspects that can affect the release approval.

[](https://www.panaya.com/wp-content/uploads/2020/10/Transparency.jpg)Taken from

[Project managers](https://www.panaya.com/blog/modern-alm/portfolio-program-management-critical-app-delivery/) have a high level view of all the components in the product—risk assessment and searches for opportunities, product features and their priority, quality, testing and defects, and the deployment of the app in production. This is so they can build a project scope, evaluate the work involved, and create a schedule. Responding to their requirements and reflecting the appropriate information can change the course of a project and lead to its success.

**Teams** should continue working through the list of information clients, recognize their part and responsibilities in the product delivery procedure, and create transparency to support the manner in which they make use of the information.

There are two more types of transparency—**upper management visibility**and**customer engaging**. Teams that want to build a reputation of success and create perception of prosperity, need to be able to share their development process, show how it improves over time, and how they learn from and fix mistakes. This can be achieved in either low or high intensity of reflection, depending on the situation. Usually, teams are happier to share and expose their process and performance only when the management is transparent as well. From strategic decisions through goals, promotions, hiring or firing, being open and communicating decisions that impact a team make the working environment more stable and invites identical behavior from the R&D teams.

Customer engagement is a doctrine of its own, but R&D teams that build a communication channel with customers create trust with the customer.

Conclusion

Transparency is a key asset for creating a successful, collaborative organization, but only a few organizations put into practice the activities required for achieving it. Teams find it hard to invest time in creating understandable visibility of their processes or progress. Some teams are anxious about sharing their status as it might indicate setbacks. Such teams usually don’t understand how reflecting status — no matter what’s going on — benefits the organization and affects the way the team is perceived. As there are quite a few tools in the industry that help teams get to high levels of visibility and clarity, it’s simple enough to get started today.

With the right methodologies and a designated platform, even [enterprise organizations can be agile](https://www.panaya.com/testing/white-papers/lp/experience-enterprise-agility/?cid=701w0000000fQZf).  Download our White Paper, [Introducing Agility to Enterprise Applications](https://www.panaya.com/testing/white-papers/lp/experience-enterprise-agility/?cid=701w0000000fQZf) to learn how you can start releasing frequent, high-quality change today.