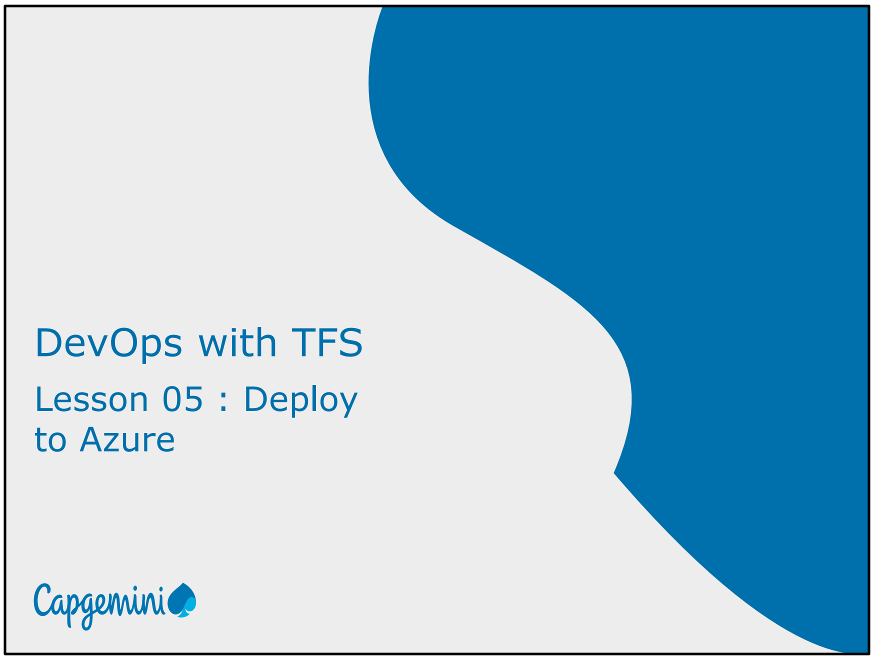
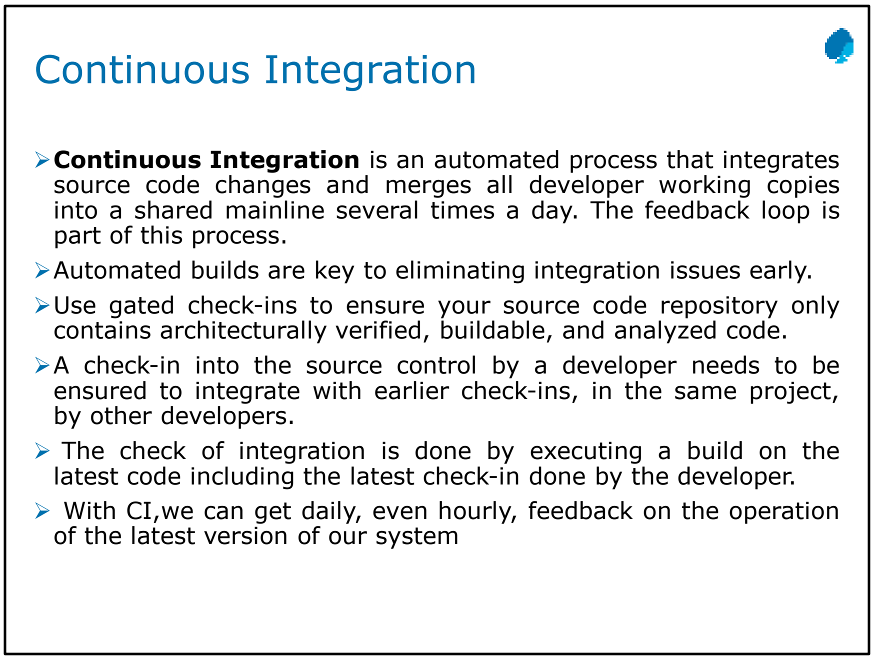


DevOps with TFS

Lesson 05 : Deploy to Azure

Capgemini

Table of Contents

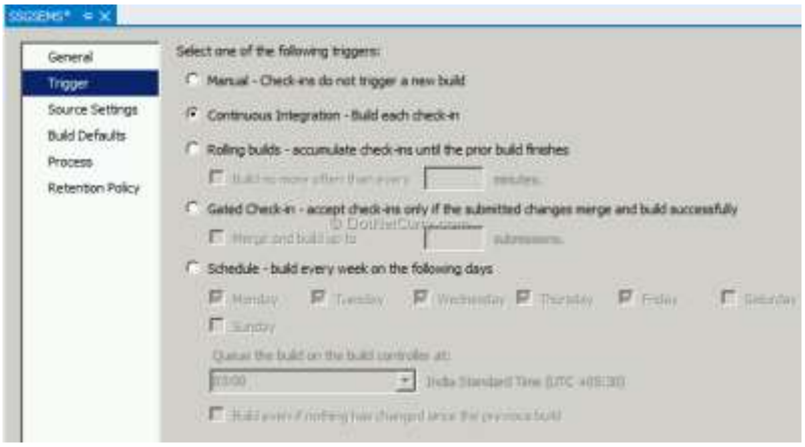


Continuous Integration

- **Continuous Integration** is an automated process that integrates source code changes and merges all developer working copies into a shared mainline several times a day. The feedback loop is part of this process.
- Automated builds are key to eliminating integration issues early.
- Use gated check-ins to ensure your source code repository only contains architecturally verified, buildable, and analyzed code.
- A check-in into the source control by a developer needs to be ensured to integrate with earlier check-ins, in the same project, by other developers.
- The check of integration is done by executing a build on the latest code including the latest check-in done by the developer.
- With CI,we can get daily, even hourly, feedback on the operation of the latest version of our system

Automated builds are key to eliminating integration issues early. Use gated check-ins to ensure your source code repository only contains architecturally verified, buildable, and analyzed code. Use Team Foundation Server or Visual Studio Online to easily define builds in an intuitive web interface using a rich gallery of reusable and extensible build tasks with support for Ant, Maven and Gradle and the ability run your build agent on Windows, Linux or the Mac.
TFS supported CI right from the beginning, when it came into existence in 2005. It has a feature for build management that includes build process management and automated triggering of the build. There are number of automated build triggers supported by TFS. Scheduled build is one of them but for us, more important is Continuous Integration trigger. It triggers a build whenever a developer checks in the code that is under the scope of that build.

Continuous Integration



Select one of the following triggers:

( ) Manual - Check-ins do not trigger a new build

[✓ Continuous Integration - Build each check-in]

( ) Rolling builds - accumulate check-ins until the prior build finishes

☐ Build no more often than every [ ] minutes.

( ) Gated Check-in - accept check-ins only if the submitted changes merge and build successfully

☐ Merge and build up to [ ] submissions.

( ) Schedule - build every week on the following days

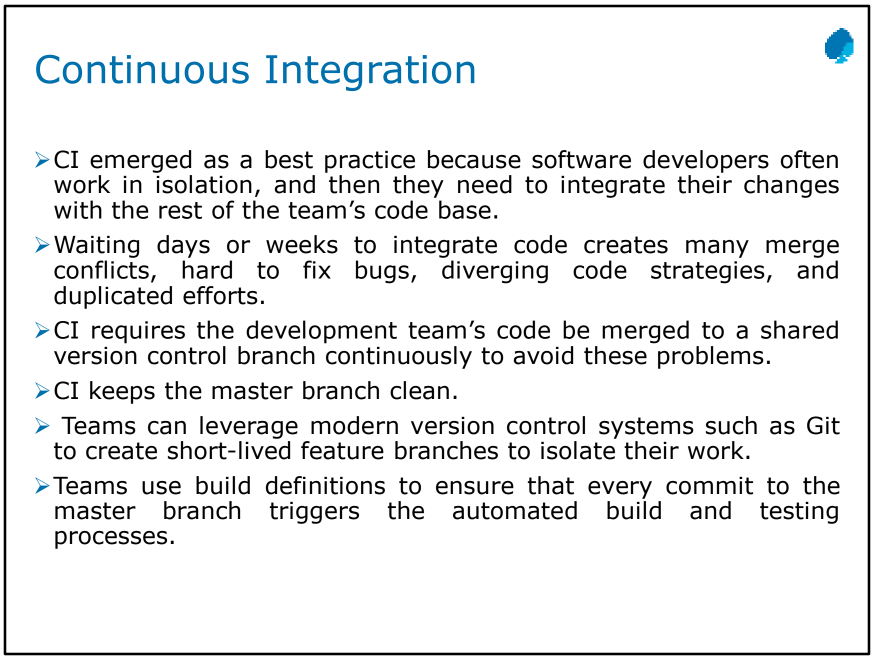
☑ Monday ☑ Tuesday ☑ Wednesday ☑ Thursday ☑ Friday ☐ Saturday

☐ Sunday

Queue the build on the build controller at:

03:00 India Standard Time (UTC +05:30)

☐ Build even if nothing has changed since the previous build



Continuous Integration

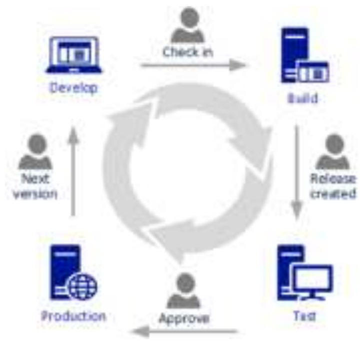
- CI emerged as a best practice because software developers often work in isolation, and then they need to integrate their changes with the rest of the team's code base.
- Waiting days or weeks to integrate code creates many merge conflicts, hard to fix bugs, diverging code strategies, and duplicated efforts.
- CI requires the development team's code be merged to a shared version control branch continuously to avoid these problems.
- CI keeps the master branch clean.
- Teams can leverage modern version control systems such as Git to create short-lived feature branches to isolate their work.
- Teams use build definitions to ensure that every commit to the master branch triggers the automated build and testing processes.

Continuous Integration (CI) is the process of automating the build and testing of code every time a team member commits changes to version control. CI encourages developers to share their code and unit tests by merging their changes into a shared version control repository after every small task completion. Committing code triggers an automated build system to grab the latest code from the shared repository and to build, test, and validate the full master branch (also known as the trunk or main).

A developer submits a "pull request" when the feature is complete and, on approval of the pull request, the changes get merged into the master branch. Then the developer can delete the previous feature branch. Development teams repeat the process for additional work. The team can establish branch policies to ensure the master branch meets desired quality criteria.

Continuous Integration

- CI ensures bugs are caught earlier in the development cycle, which makes them less expensive to fix.
- Automated tests run for every build to ensure builds maintain a consistent quality.



Overview

Welcome

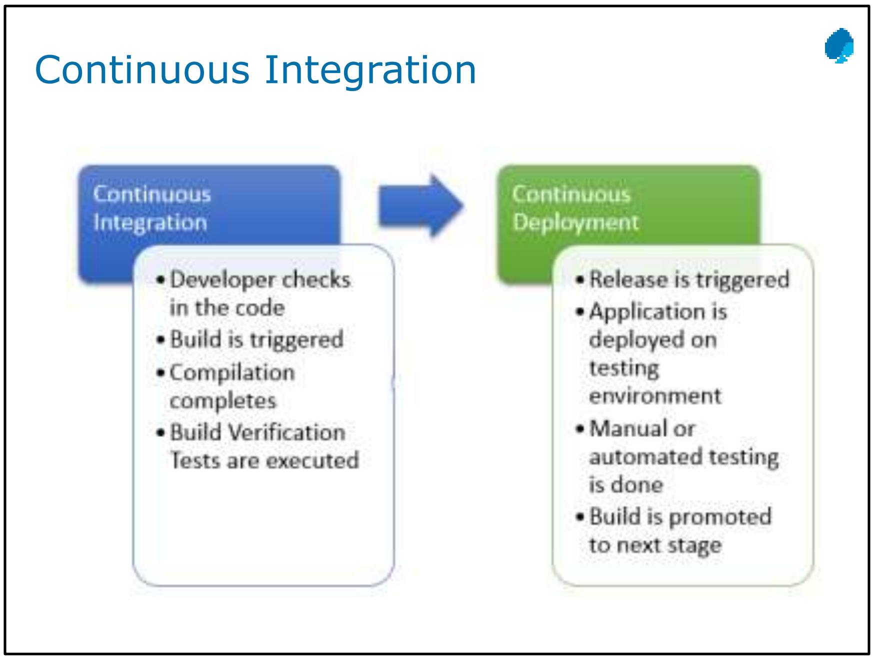
Get started using Visual Studio Team Services to make the most of your team dashboard.

Manage Work
Add work to your board

Collaborate on code
Add code to your repository

Continuously integrate
Automate your builds

Visualize progress
Learn how to add charts



Continuous Integration

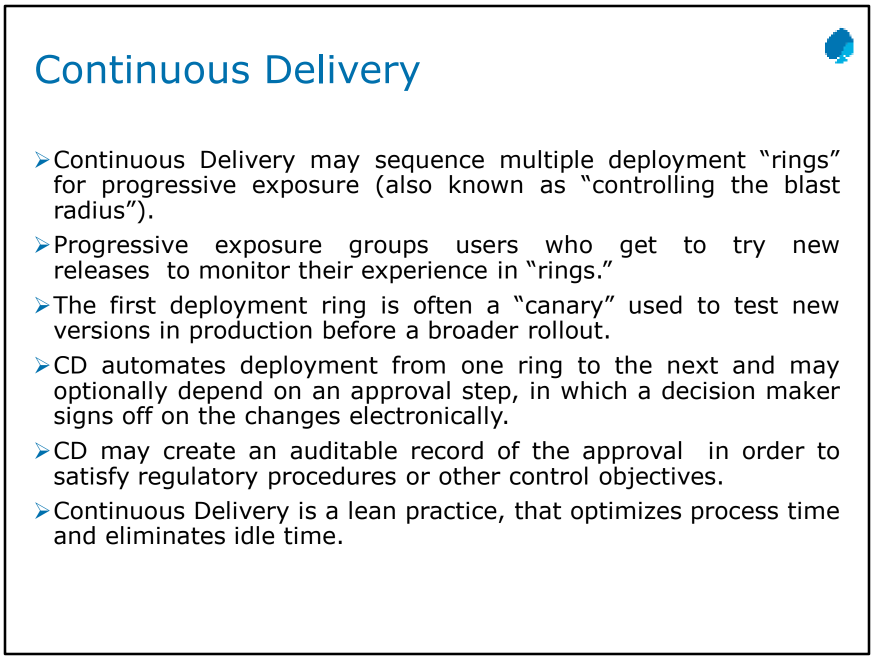
- Developer checks in the code
- Build is triggered
- Compilation completes
- Build Verification Tests are executed

- Release is triggered
- Application is deployed on testing environment
- Manual or automated testing is done
- Build is promoted to next stage

You're in trouble if you find any of the following in your SDLC:
You're deploying your applications manually, it's a pain in the neck and it takes a lot of time.
You manually perform source code integrations, and conflicts often occur.
You manually edit application configuration in staging or production environments.
You tell other developers that "it works on my machine" and the problem is in their code.
You have automated tests, but don't rely on them.
You often have a last-minute hotfix in production.
You feel like you work as a "fireman" more often than as an engineer.

Continuous Delivery

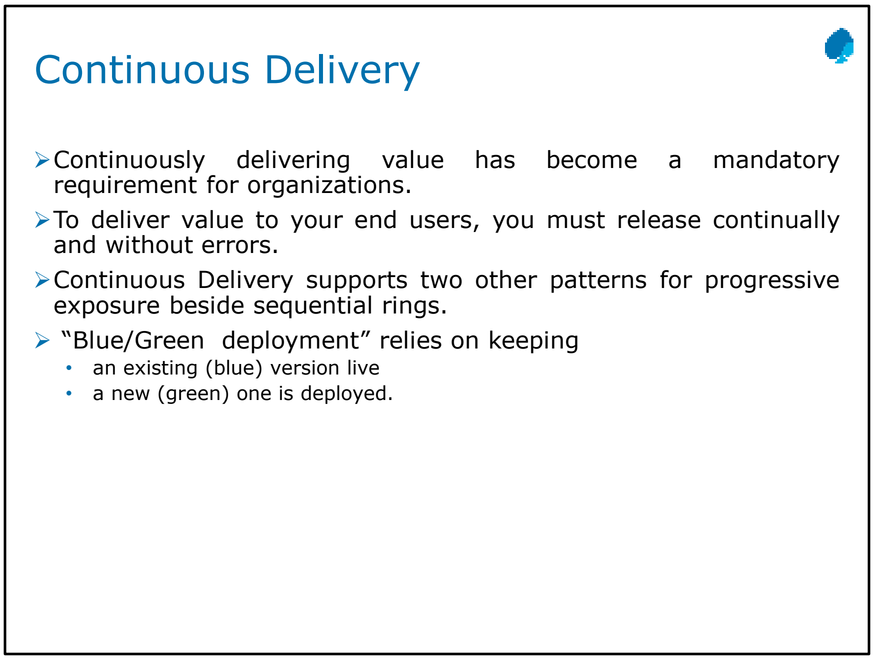
- Continuous Delivery means starting an automated deployment process whenever a new successful build is available.
- Continuous Delivery (CD) is the process to build, test, configure and deploy from a build to a production environment.
- Multiple testing or staging environments create a *Release Pipeline* to automate the creation of infrastructure and deployment of a new.
- Successive environments support progressively longer-running activities of integration, load, and user acceptance testing.
- Continuous Integration starts the CD process and the pipeline stages each successive environment the next upon successful completion of tests.



Continuous Delivery

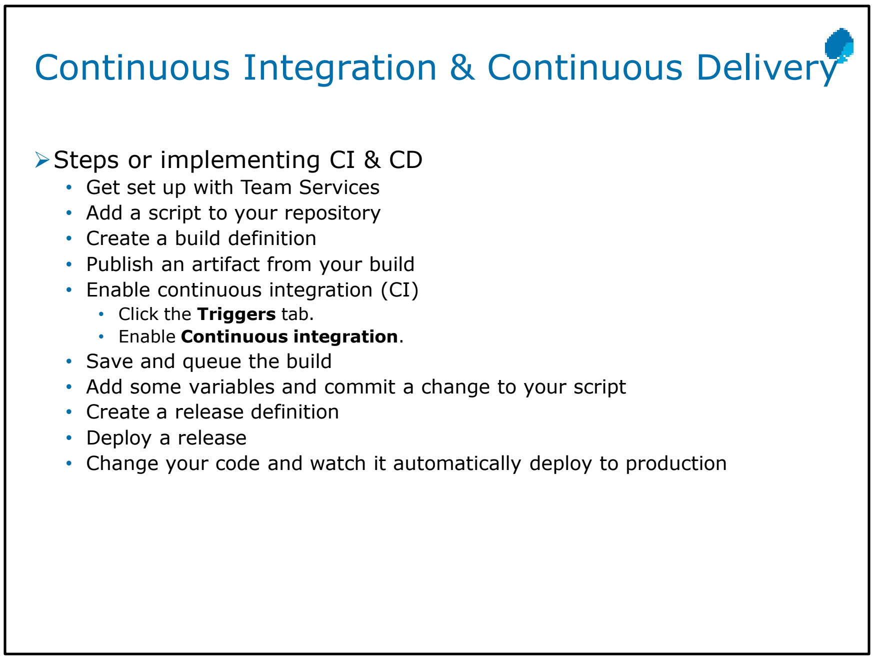
- Continuous Delivery may sequence multiple deployment "rings" for progressive exposure (also known as "controlling the blast radius").
- Progressive exposure groups users who get to try new releases to monitor their experience in "rings."
- The first deployment ring is often a "canary" used to test new versions in production before a broader rollout.
- CD automates deployment from one ring to the next and may optionally depend on an approval step, in which a decision maker signs off on the changes electronically.
- CD may create an auditable record of the approval in order to satisfy regulatory procedures or other control objectives.
- Continuous Delivery is a lean practice, that optimizes process time and eliminates idle time.

Without Continuous Delivery, software release cycles were previously a bottleneck for application and operation teams. Manual processes led to unreliable releases that produced delays and errors. These teams often relied on handoffs that resulted in issues during release cycles. The automated release pipeline allows a "fail fast" approach to validation, where the tests most likely to fail quickly are run first and longer-running tests happen after the faster ones complete successfully. Continuous Delivery is a lean practice. The goal of CD is to keep production fresh by achieving the shortest path from the availability of new code in version control or new components in package management to deployment. By automation, CD minimizes the time to deploy and *time to mitigate* or *time to remediate* production incidents (TTM and TTR). In lean terms, this optimizes process time and eliminates idle time.



Continuous Delivery

- Continuously delivering value has become a mandatory requirement for organizations.
- To deliver value to your end users, you must release continually and without errors.
- Continuous Delivery supports two other patterns for progressive exposure beside sequential rings.
- "Blue/Green deployment" relies on keeping
  - an existing (blue) version live
  - a new (green) one is deployed.

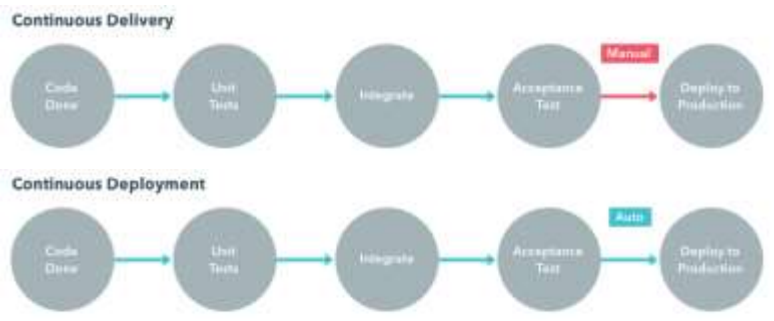


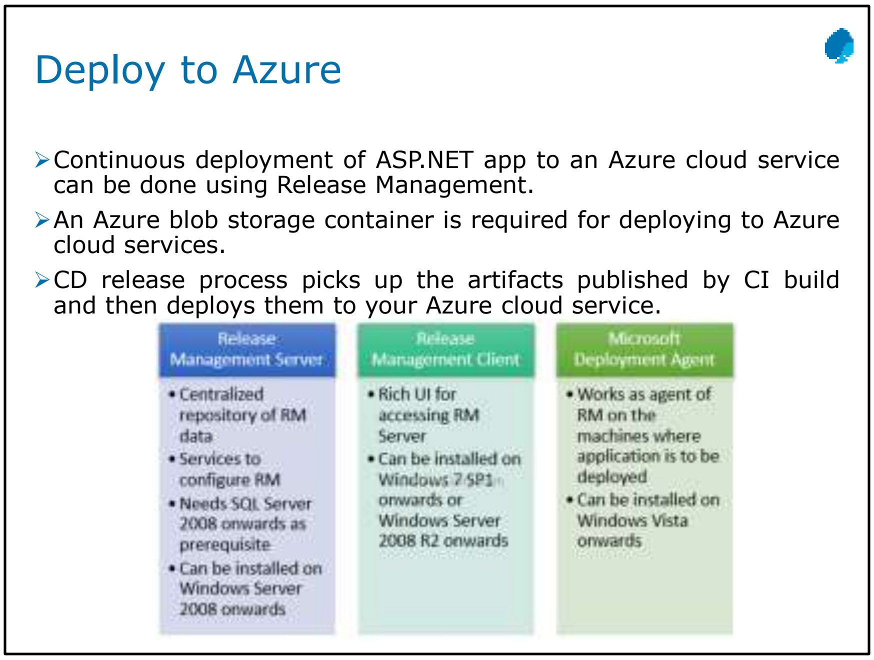
Continuous Integration & Continuous Delivery

- Steps or implementing CI & CD
  - Get set up with Team Services
  - Add a script to your repository
  - Create a build definition
  - Publish an artifact from your build
  - Enable continuous integration (CI)
    - Click the **Triggers** tab.
    - Enable **Continuous integration**.
  - Save and queue the build
  - Add some variables and commit a change to your script
  - Create a release definition
  - Deploy a release
  - Change your code and watch it automatically deploy to production

Continuous Deployment

- Continuous deployment means starting an automated deployment process whenever a new successful build is available.
- **Continuous Deployment** is an extended Continuous Delivery process with additional steps automating deploying to staging and production.



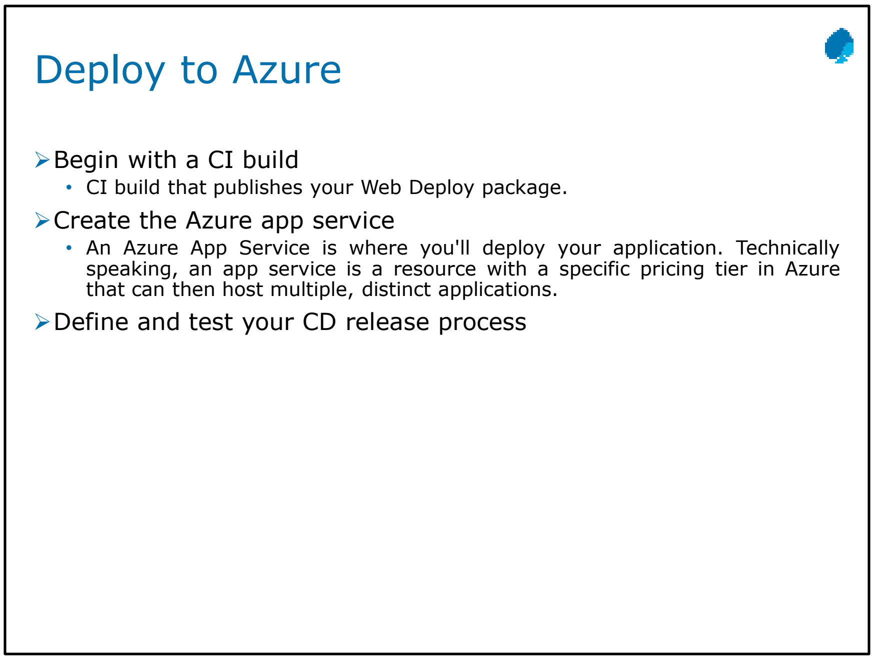


Deploy to Azure

- ➢ Continuous deployment of ASP.NET app to an Azure cloud service can be done using Release Management.
- ➢ An Azure blob storage container is required for deploying to Azure cloud services.
- ➢ CD release process picks up the artifacts published by CI build and then deploys them to your Azure cloud service.

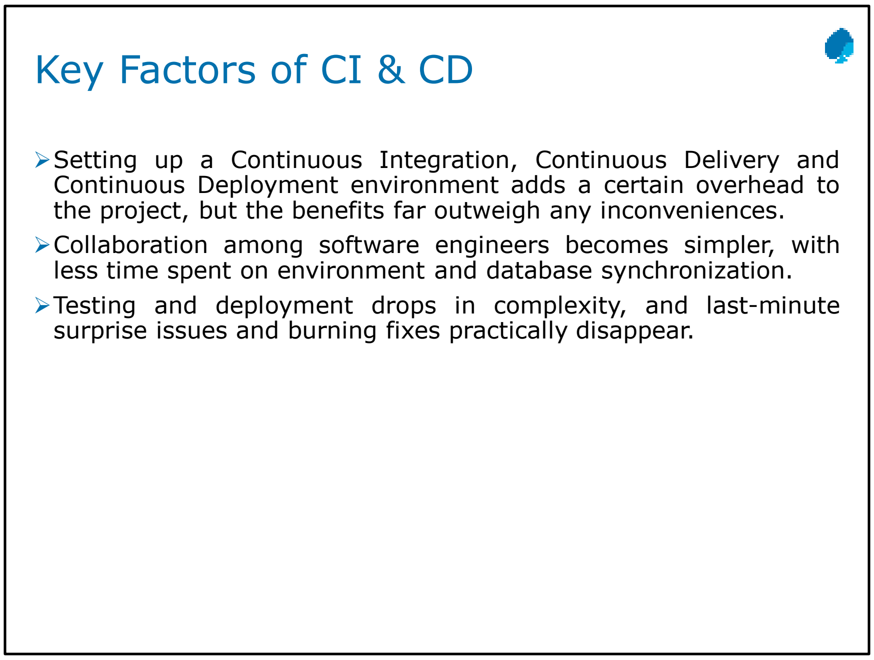
Release Management Server	Release Management Client	Microsoft Deployment Agent
• Centralized repository of RM data • Services to configure RM • Needs SQL Server 2008 onwards as prerequisite • Can be installed on Windows Server 2008 onwards	• Rich UI for accessing RM Server • Can be installed on Windows 7 SP1 onwards or Windows Server 2008 R2 onwards	• Works as agent of RM on the machines where application is to be deployed • Can be installed on Windows Vista onwards

Begin with a CI build



Deploy to Azure

- Begin with a CI build
  - CI build that publishes your Web Deploy package.
- Create the Azure app service
  - An Azure App Service is where you'll deploy your application. Technically speaking, an app service is a resource with a specific pricing tier in Azure that can then host multiple, distinct applications.
- Define and test your CD release process



Key Factors of CI & CD

- Setting up a Continuous Integration, Continuous Delivery and Continuous Deployment environment adds a certain overhead to the project, but the benefits far outweigh any inconveniences.
- Collaboration among software engineers becomes simpler, with less time spent on environment and database synchronization.
- Testing and deployment drops in complexity, and last-minute surprise issues and burning fixes practically disappear.

DevOps - Best Practices

- Always use a version control system.
- Automate the build.
- Your build should be self-testing
- Commit at least every day.
- Keep the build fast.
- Test in a staging environment before deploying software on production.
- Make it easy for everyone to get the latest executable.
- Everyone should see what's happening.
- Prepare environments.

Summary

- ➢Understanding Continuous Integration in VSTS
- ➢ How CI is connected to CD.
- ➢ Steps involved in CI & CD
- ➢ Advantages of CI & CD
- ➢Deploying to Azure
- ➢ Best Practices to be followed