The Business Case
Set Yourself Up for Success in Three Months
Jeffrey Palermo
Jeffrey@clear-measure.com
@jeffreypalermo

Chief Architect &
CEO, Clear
Measure, Inc.

Author of
“.NET DevOps for Azure”
aka.ms/dotnetdevopsbook

13-time Microsoft
MVP award

Host of the “Azure
DevOps Podcast”
Complete DevOps environment
“Pit of Success” configuration
.NET Core, SQL Database, Azure, IaC
Unit tests, Selenium tests, and more
Jumpstart your DevOps efforts – clone the project
compiled by Jeffrey Palermo
Today and Week 1
Plan and Pitch

First Month
Initiate Changes in a Proof of Concept (POC)

Months 2-3
Map Process Changes and Land Your POC
Today and Week 1
Plan and Pitch
The Three “Ways” of DevOps

Flow
Integrate & automate

Feedback
Database deployments

Protect
And preserve data

Continuously Learn
Monitor & analyze
<table>
<thead>
<tr>
<th>Conceptual Definition</th>
<th>User Experience Design</th>
<th>Technical Design</th>
<th>Test Design</th>
<th>Development</th>
<th>Functional Validation</th>
<th>User Experience Validation</th>
<th>Release Validation</th>
<th>Release (Done)</th>
</tr>
</thead>
</table>
Developed by many

Team

Has many

Has many

Segmented into many

Software System

Resides in one

Owned by one

Git Repository

Has one

Versioned Software Application

Builds one

Has one

Software Build Configuration
Standardizing source code (while deploying manually)

**Pros**
- Provides audit trail of development changes
- Manages collaboration of team members
- Standardizes coding practices

**Cons**
- Manual database code validation, testing, and deployments
- Requires a custom process to prevent environment drift
- Failed deployments and downtime
Automation adds these benefits:

1. Code validation and static analysis
2. Speeds up deployments with continuous delivery
3. Prevents environment drift
<table>
<thead>
<tr>
<th>Conceptual Definition</th>
<th>User Experience Design</th>
<th>Technical Design</th>
<th>Test Design</th>
<th>Development</th>
<th>Functional Validation</th>
<th>User Experience Validation</th>
<th>Release Validation</th>
<th>Release (Done)</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Diagram" /></td>
<td><img src="image2.png" alt="Diagram" /></td>
<td><img src="image3.png" alt="Diagram" /></td>
<td><img src="image4.png" alt="Diagram" /></td>
<td><img src="image5.png" alt="Diagram" /></td>
<td><img src="image6.png" alt="Diagram" /></td>
<td><img src="image7.png" alt="Diagram" /></td>
<td><img src="image8.png" alt="Diagram" /></td>
<td><img src="image9.png" alt="Diagram" /></td>
</tr>
</tbody>
</table>
SQL Provision adds speed & safety

Rapidly refresh development databases with real-world datasets

Easily support dedicated environments that isolate changes made in branches and allow experimentation

Mask sensitive data upon image creation
<table>
<thead>
<tr>
<th>Environment Attribute</th>
<th>TDD</th>
<th>UAT</th>
<th>PROD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose</td>
<td>Automated Verification</td>
<td>Manual Verification</td>
<td>Ongoing Operations</td>
</tr>
<tr>
<td>Audience</td>
<td>Engineering Team</td>
<td>Internal Customers</td>
<td>External Customers</td>
</tr>
<tr>
<td>Quality Control</td>
<td>Acceptance Testing</td>
<td>UX Testing</td>
<td>Automated Alarms</td>
</tr>
<tr>
<td></td>
<td>Env. Recreation</td>
<td>Exploratory Testing</td>
<td>Health Checks</td>
</tr>
<tr>
<td></td>
<td>Health Checks</td>
<td>Database Migration</td>
<td>Tracer Bullets</td>
</tr>
<tr>
<td></td>
<td>Database Recreation</td>
<td>Health Checks</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Security Testing</td>
<td>Observability Validation</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Scale/Perf Testing</td>
<td>Other Manual Testing</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Reliability Testing</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Other Test Suites</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Environment Deploy Steps</td>
<td>TDD</td>
<td>UAT</td>
<td>PROD</td>
</tr>
<tr>
<td>--------------------------------------------------------------</td>
<td>-----</td>
<td>-----</td>
<td>------</td>
</tr>
<tr>
<td>Migrate Database</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Deploy Application</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Run Health Check</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Load Static Data</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Load Test Data</td>
<td>✓</td>
<td>On-Demand</td>
<td>✓</td>
</tr>
<tr>
<td>Recreate Environment</td>
<td>✓</td>
<td>On-Demand</td>
<td>✓</td>
</tr>
<tr>
<td>Recreate Database</td>
<td>✓</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Run Acceptance Tests</td>
<td>✓</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Deploy Unattended</td>
<td>✓</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Destroys Environment</td>
<td>✓</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>
Should monitoring be in your initial project?

**Pros**
- Visibility into production health
- SQL Monitor shows when deployments have occurred, making it easy to track customer impact (good and bad) of changes

**Cons**
- Shines light on problems that might have been present for quite a while
Think about your company’s values and goals

- Excellent customer experience
- Faster time to market
- Culture of innovation
- More productive use of DBA and developer time
- Great employee retention
- Lower risk of data breaches
“We should standardize database code and automate deployments to reduce feature time to market.”
“We should automate database DevOps to solve our hectic and disruptive deployments”
“We will have no hope of modernizing our large system unless we automate our database environments”
First Month
Initiate Changes in a Proof of Concept (POC)
In a supported POC, start small

Integrate tools & infrastructure

Select a small system/database

Limit dependencies
The goal of a Proof of Concept is demonstrating that the tools achieve your success criteria.
**Steps**

- Private build should run automated unit tests and component level integration tests on a local workstation before check-in
- No more than 10 minutes (<5 preferable)

**Private build**

- Everything that belongs to the application stored in source control: Ex: database schema scripts
- Nothing that belongs to environments is stored in source control: ex: credentials, environment names, etc

- Developer checks in code
- Detects source change
- Package Mgmt
  - Azure Artifacts or OD’s built-in Nuget server
  - Uses standard *.nupkg package format
  - Retains release candidates so they are always available for deployment

- Azure Pipelines Release Hub orchestrates the deployment pipeline
- All logs of every step in a deployment are recorded
- Security is managed at this level
- Multiple pipelines can be modelled

- Azure Pipelines
- Monitors for source changes
- Houses all source code and work tracking
- Provides dashboards and reports of builds and deployments
- Keeps all records of test runs and reports
- Provides full availability of what user checked in what code and what user approved what deployment

- Azure Pipelines
- Runs all steps in private build
- Embeds software version number into .Net assemblies
- Performs static code analysis validation
- Produces the release candidate packages w/ OctoPack (one package per major application component)
- Note: Full system acceptance test harnesses should be packaged as well since they will need to be deployed to and run on a downstream environment

- Azure DevOps Services
- This class of environment can be 2 or more environments depending on # of levels of verification needed
- First environment for human testing
- Often used for UAT or Staging purposes
- Uses a full-size production-like data set or redacted production backup
- Updates state (database, files) while preserving data
- Post-deployment, can run performance & other non-functional (security) tests
- Configures environment at same scale as production
- Manual testing is performed here

- Octopus Deployment
- First deployed environment
- Configures environment upon deployment
- Rebuilds database and test data each time
- Runs full system acceptance tests
- Runs other test suites needing a fully deployed environment
- No humans on this environment – it is fully automated and constantly refreshing
- This environment deploys builds from all branches, including feature branches (so we know they fully work)

- DevOps pipeline
- Recreate database
- Initialize database
- Configure Server/Resources
- Deploy application
- Run acceptance tests (Azure Pipelines)

- Deployment Ring (if applicable)

- First Line TID Environment Deployment
  - Managed Octopus Deploy
  - Managed Octopus Deploy
  - Managed Octopus Deploy

- 2nd Line Manual (or UAT) Environment Deployment
  - Rebuilds database and test data each time
  - Runs full system acceptance tests
  - Runs other test suites needing a fully deployed environment
  - No humans on this environment – it is fully automated and constantly refreshing
  - This environment deploys builds from all branches, including feature branches (so we know they fully work)

- DevOps pipeline
- Ensure backups
- Migrate database schema
- Configure Server/Resources
- Deploy application
- Application smoke test

- Post-deployment, can run performance & other non-functional (security) tests
- Configures environment at same scale as production
- Manual testing is performed here

- Managed "maintenance" page if a rolling deployment is not used

- 2nd Line UAT Environment Deployment
  - Rebuilds database and test data each time
  - Runs full system acceptance tests
  - Runs other test suites needing a fully deployed environment
  - No humans on this environment – it is fully automated and constantly refreshing
  - This environment deploys builds from all branches, including feature branches (so we know they fully work)

- DevOps pipeline
- Ensure backups
- Migrate database schema
- Configure Server/Resources
- Deploy application
- Application smoke test

- Post-deployment, can run performance & other non-functional (security) tests
- Configures environment at same scale as production
- Manual testing is performed here

- Managed "maintenance" page if a rolling deployment is not used

- Production Environment Deployment
  - Managed Octopus Deploy
  - Managed Octopus Deploy
  - Managed Octopus Deploy
Developer checks in code

Azure Repos Git VCS

- Everything that belongs to the application stored in source control. Ex: database schema scripts
- Nothing that belongs to environments is stored in source control: ex: credentials, environment names, etc

Azure DevOps Services

- Monitors for source changes
- Houses all source code and work tracking
- Provides dashboards and reports of builds and deployments
- Keeps all records of test runs and reports
- Provides full auditability of what user checked in what code and what user approved what deployment

Continuous Integration Build

- Runs all steps in private build
- Embeds software version number into .Net assemblies
- Performs static code analysis validation
- Produces the release candidate packages w/ OctoPack (one package per major application component)
- Note: Full system acceptance test harnesses should be packaged as well since they will need to be deployed to and run on a downstream environment

Package Mgmt

- Azure Artifacts or OD’s built-in Nuget server
- Uses standard *.nupkg package format
- Retains release candidates so they are always available for deployment

Developer checks in code

- Private build should run automated unit tests and component-level integration tests on a local workstation before check-in
- No more than 10 minutes (<5 preferrable)
- First deployed environment
- Configures environment upon deployment
- Rebuilds database and test data each time
- Runs full system acceptance tests
- Runs other test suites needing a fully deployed environment
- No humans on this environment – it is fully automated and constantly refreshing
- This environment deploys builds from all branches, including feature branches (so we know they fully work)

- This class of environment can be 2 or more environments depending on # of levels of verification needed
- First environment for human testing
- Often used for UAT or Staging purposes
- Uses a full-size production-like data set or redacted production backup
- Updates state (database, files) while preserving data
- Post-deployment, can run performance & other non-functional (security) tests
- Configures environment at same scale as production
- Manual testing is performed here

- Same deployment process as Staging/2nd line
- Includes explicit and audited gated approval before commencing deployment
- Abbreviated post-deploy smoke test to ensure application starts up properly
- Managed “maintenance” page if a rolling deployment is not used

- Recreate database
- Initialize database
- Configure Server/Resources
- Deploy application
- Run acceptance tests (Azure Pipeline)

- Ensure backups
- Migrate database schema
- Configure Server/Resources
- Deploy application
- Application smoke test

- Ensure backups
- Migrate database schema
- Configure Server/Resources
- Deploy application
- Application smoke test
Private Build

Version Control

- Everything that belongs to the application stored in source control. Ex: database schema scripts
- Nothing that belongs to environments is stored in source control: ex: credentials, environment names, etc

| • Private build should run automated unit tests and component-level integration tests on a local workstation before check-in
| • No more than 10 minutes (<5 preferable) |

CI Build

Continuous Integration Build

| • Runs all steps in private build
| • Embeds software version number into .Net assemblies
| • Performs static code analysis validation
| • Produces the release candidate packages (one package per major application component)
| • Note: Full system acceptance test harnesses should be packaged as well since they will need to be deployed to and run on a downstream environment |

Release Candidates

Package Mgmt

| • Azure Artifacts or OD’s built-in Nuget server
| • Uses standard *.nupkg package format
| • Retains release candidates so they are always available for deployment |

Full-System Tests

Run tests on Automated Dev Environment

| • First deployed environment
| • Configures environment upon deployment
| • Rebuilds database and test data each time
| • Runs full system acceptance tests
| • Runs other test suites needing a fully deployed environment
| • No humans on this environment – it is fully automated and constantly refreshing
| • This environment deploys builds from all branches, including feature branches (so we know they fully work) |
Months 2-3
Map Process Changes and Land Your POC
A POC doesn’t take 2 months of sustained work

The two-three month timeline applies if:
  • Your internal point team has other duties
  • You only get a few hours of their time each week

Dedicated teams have completed Database DevOps POCs in two weeks
What happens when a build fails, and how do we respond?

When will the risk of changes be assessed? Pull requests? Release artifacts?

Can security assessments be moved into checklists?

How often will deployments to databases be done?

How many changes should be shipped in a single deployment?

What approval processes are best for deployments?
Shifting change approval left

Shifting change review and approval to earlier in the development cycle speeds you up

First steps: can standard (pre-approved) changes help get you started?

Sponsorship: this is one place where a sponsor who can help you build a coalition will help
<table>
<thead>
<tr>
<th>Conceptual Definition</th>
<th>User Experience Design</th>
<th>Technical Design</th>
<th>Test Design</th>
<th>Development</th>
<th>Functional Validation</th>
<th>User Experience Validation</th>
<th>Release Validation</th>
<th>Release (Done)</th>
</tr>
</thead>
</table>
Completing your Proof of Concept

- Collate results from POC and implementation plan
- Finalize licensing
- Set target roll-out date and schedule working session to plan implementation
- Record video demo of change->deploy cycle to showcase automation and speed
When an executive sponsor has been involved from the start, POC sign-off and completion goes very smoothly.
Any questions?

My contact info:
@jeffreypalermo
Jeffrey@clear-measure.com
www.clear-measure.com