# Chapter 1: Introduction

You, dear reader, are starting down the path of excellence. By picking up this book, you are showing your leadership and resolve to equip your development organization to be world-class, competing with any other development group on the planet. You are taking initiative. You are a software leader. You are confronting the challenge head-on. This book is for you. This book is a synthesis of practices, tools, and process that, together, can equip a software organization to move fast and deliver software of the highest quality. In this chapter, we cover the relevant common problem our industry faces, the solution to that problem, and how to implement it for your team. This text goes hand-in-hand with a fully implemented example publicly available at https://dev.azure.com/clearmeasurelabs/Onion-DevOps-Architecture.

## The problem

Every day, millions of developers use .Net to build and operate mission-critical software systems for organizations around the world. Visual Studio, .Net, and Windows Server, whether on-premise or in Azure, provide astounding capabilities that enable any kind of software. The marketplace has scores of books, online courses, and tutorials teaching every technology framework and language feature. Microsoft’s own online documentation is broad and comprehensive. The Microsoft platform, along with the marketplaces, extensions, and packages has a building block for everything you can imagine. BUT, it is completely up to you to put it all the blocks together in just the right way for YOUR environment. This book seeks to change that.

### The challenge of explosive growth

By any measure, the number of professional developers has exploded over the last decade, surpassing the growth rate of the previous decade. In Stack Overflow’s 2018 survey, we can see that over half of professional developers entered the industry in the last five years.

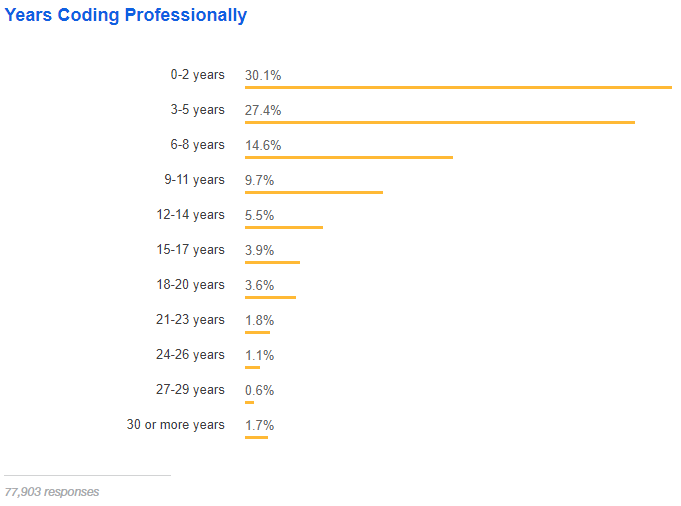


Figure: Over half of developers have five years of professional coding experience or less – Stack Overflow

If you have surpassed 20 years of development experience, you qualify at the top 5.2% of experience in the industry. We can see the inflection point between the 6-8 years and 3-5 years of experience. It’s unclear how the growth will continue or if business demand will start to be saturated. Regardless of future growth, we have a challenge in our industry created by this explosive growth in the workforce. Consider this analogy. You are opening a new auto mechanic shop.



Figure: An auto mechanic shop must be set up in a way conducive to delivering quality and speed.

You purchase a fantastic location on a main road close to other centers of business. You spare no expense building the shop. You contact a local mechanic trade school and declare you wish to hire the top 7% of up-coming graduating class. You have budgeted for whatever pay it takes to hire the best and the brightest who have just been trained as new mechanics.

Along comes graduation day, and the next week, you are preparing to open for business. You gather in front of the shop next to pallets of just-delivery tools and shop equipment. You brief your new workforce expressing excitement saying, “Let’s get the shop ready for opening. We start serving customers next week.” Your staff’s excitement turns to fear with wide eyes. Your grand opening is a disaster, and you wonder what you missed.

This manager hired staff who had been trained in how to fix and service automobiles. They were smart, skillful and motivated. They were trained in an environment that was expertly configured. Alas, the curriculum did not include how to set up a new environment for themselves.

Unfortunately, more than a few team leaders and managers have experienced a similar situation. These teams have developers who know how to apply their training and practice. But in every trade that builds something, the jobsite, or environment, has a profound effect on the effectiveness of the team. The common curriculum has some gaps, and this is one of them. This book seeks to fill that gap in the curriculum. This book will equip you to build a highly effective DevOps environment for your team.

### No end-to-end reference implementation

Along with documentation, open source projects, and samples, there does not exist any available end-to-end demonstration of a complete DevOps environment. Many have sought such a reference implementation. With a reference implementation, teams could emulate the patterns demonstrated and perform configuration rather than research and development in this area. And while tutorials and online videos exist, they demonstrate part of the solution and don’t provide access to any functional implementation that can be evaluated and copied. This book changes that.

## The solution

This book provides the model for a world-class DevOps environment when working with Microsoft technologies. And while variances in tools, language, or requirements will change the needed implementation, the DevOps model shared in this text is the architecture for the working environment for your team. Modify parts of it as you see fit, but the architecture will enable all your teams and all your applications to accelerate in performance and push forward through the next decade.

Over the past 13 years of a 22+ year career in software engineering, this author has sought to synthesize research, patterns, methods, processes, and tools that would yield the best environment for development teams. Through early Agile transformations including Scrum, Extreme Programming, Kanban, Lean Software, and other methodologies, we can see that methodology alone with not guarantee software success. In addition, tools alone cannot remove all risk from a project. Only by combining the best elements of everything available can we create an environment where it is harder to fail than success. The DevOps environment described in this book seeks to pull good ideas from all available prior art, combining them in a unique way that any team can implement. It is this author’s belief that any team can be WORLD-CLASS when equipped with the right environment, tools, and process.

This book is written with a development leader in mind. Whether you are a software architect, lead engineer, manager, executive, or a passionate leader within a development team, this book is written so that you can take action to serve your team by equipping them for success. Our target reader is one who works to enable others to be productive in shipping great software that delights your customers. To start down this path, let’s first cover the architecture of a DevOps environment.

### DevOps architecture

When designing software, or anything for that matter, one must draw what is to be built. In the software world, our best method for illustrating a software-related software architecture is Philippe Kruchten’s 4+1 model. He writes about it in his IEEE paper, “Architectural Blueprints—The “4+1” View Model of Software Architecture”[[1]](#footnote-1). In this model for emergent and iterative architecture, Mr. Kruchten defines four layers to illustrate the architecture and one list of scenarios, which are select use cases the architecture supports. The needed capabilities drive the architecture. Just like blueprints for a house will have layers such as floorplan, electrical, and plumbing, the four layers in this model for software architecture are:

* Logical view
* Process view
* Physical view
* Development view

#### Logical view

Let’s consider the very popular logo that has come to represent DevOps. This figure us taken from the Wikimedia Commons site[[2]](#footnote-2).

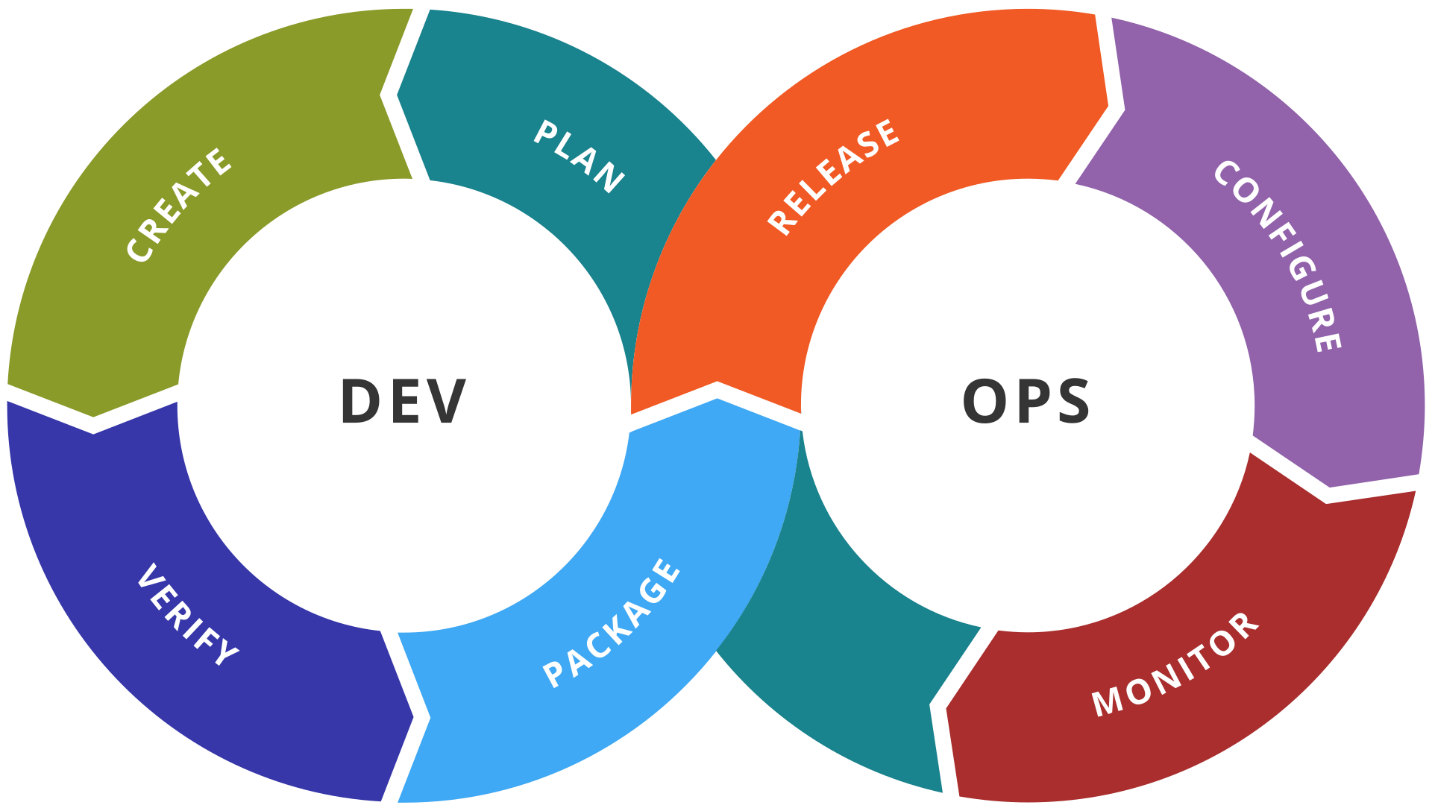


Figure: The common logo for DevOps still does not combine Dev and Ops

There are many versions of this same depiction of DevOps, but this author believes it is fundamentally wrong. It fails to unify Development and Operations. It still maintains that they are two. And if there are two that must work well together, then one would question: “where is product management” or “where is QA”. And once these other groups are depicted, we have an organization that still hangs onto the silos. No, a much better logical view of DevOps is the following.

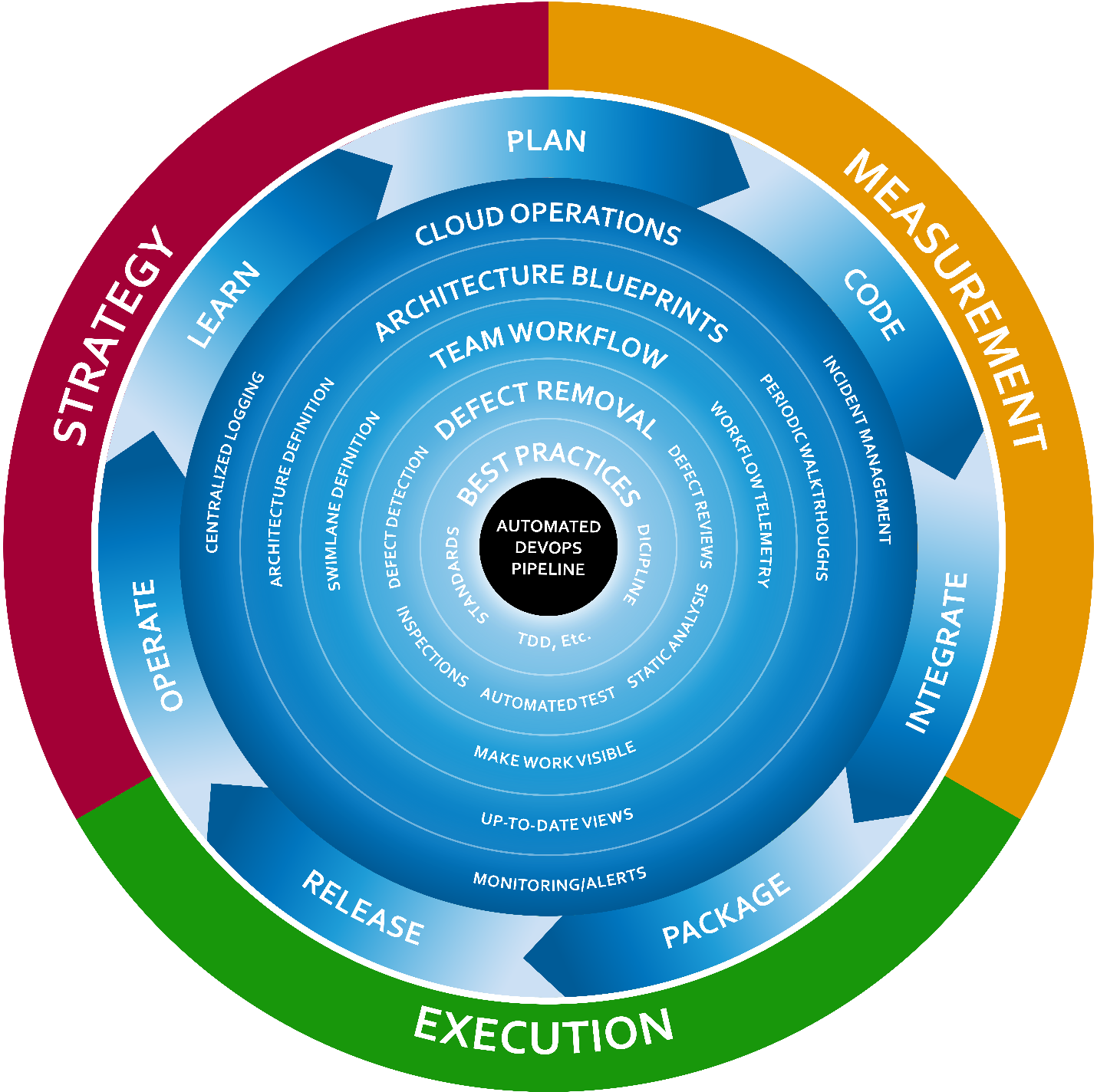


Figure: Logically, DevOps seeks to drive a fast cycle time with a unified organization

A more appropriate view of DevOps would untwist the sideways figure eight into a unified cycle. The goal of the unified team is to complete a cycle in as short a time as possible. The cycle includes:

* Plan
* Code
* Integrate
* Package
* Release
* Operate
* Learn

In order to facilitate a constant acceleration of this cycle, we define a hub around which this cycle can spin. That hub is the automated DevOps pipeline. Built on top of that foundation, we also have several layers of capability to achieve in our team. Each layer needs:

* Strategy: decisions on what is to be done and how
* Execution: competent and faithful implementation of the strategy on an ongoing basis
* Measurement: inspecting and verifying that executing the strategy is achieving the desired objectives

The six layers of capability for the team in this model are:

* Automated DevOps Pipeline: an automated way to convert code into production software
* Best Practices: selection and appropriate implementation of the practices that are deemed to be the best for the software and the team’s situation
* Defect Removal: Choice of defect prevention and defect removal techniques and the application thereof
* Team Workflow: Complete visibility into all the work the team is doing with the ability to see bottlenecks quickly
* Architectural Blueprints: the definition, maintenance, and inspection of clear blueprints for the software as is and as the software is to be in the next increment
* Cloud Operations: How the software is being operated, monitored, and customers supported in production

This is the logical model for a team’s DevOps environment. As you evaluate these layers, you will not find a single cookie-cutter implementation that is right for every team, but each of these layers of capability must be addressed for every team. As you are analyzing this model for you team, don’t hesitate to add an additional layer if your context deems that appropriate.

Remember that the most important element of this logical architecture is the cycle. Everything is subordinate to the ability to continually accelerate the team’s ability to cycle of planning an idea to learning that the intended outcome had been accomplished. Only about 1/3 of ideas that are prioritized for a software system end up having the positive affect that is intended. Sam Guckenheimer, Product Owner for the Azure DevOps product line at Microsoft, has shared his analysis of relevant industry research in a 2018 podcast interview[[3]](#footnote-3). At 15:43 of the interview, Mr. Guckenheimer relates the “rule of thirds” whereby

* One third of prioritized features have the positive, intended effect
* One third of prioritized features have a neutral effect
* One third of prioritized features have a negative effect and should be reverted immediately.

If even the best, most sophisticated companies are still subject to this general rule, it is imperative that a software organization be able to execute a software cycle very quickly. Companies that can drive cycle time lower will have a sustained competitive advantage in the marketplace.

#### Process view

In 4+1 architecture, the process view follows the order in which things are done. This view, and the structure thereof will guide many other implementation decisions as tools are integrated and methods chosen. The following is an end-to-end process view for the model of DevOps that will be illustrated throughout this book.



Figure: The DevOps’ architecture process view

The process for a DevOps environment contains more than just automated builds and deployments. It starts by modeling the entire value chain from the time an idea is being discussed to when that idea has been put into the hands of customers as a new software capability. Before code is even touched, there are four distinct types of design that must be performed on a feature so that developers know how it should be implemented. Some small teams do not track their process to this level of granularity. Instead, they rely on conversations with a Product Owner, as in Scrum, to have questions answered. This can work fine at small scale, but a high-performing process enumerates every distinct type of work that must be performed and separates them in sequence. This allows for the measurement of work in process (WIP) and throughput of each workstation. The modern DevOps books that will be cited in this book all credit their thinking on the concept of flow to Eliyahu Goldratt, author of “The Goal: A Process of Ongoing Improvement”[[4]](#footnote-4). This is the same author of the popular book, “Theory of Constraints”[[5]](#footnote-5). This concept of flow, as within a manufacturing plant, has us design the process so that we can visualize the amount of work at any given phase of work and make sure none of them become a bottleneck in the overall process. One step will mathematically always be the bottleneck, so our continual process improvements in search of an ever-quicker cycle time will be targeted at the phase of work that is the currently holding up further rates of throughput.

#### Physical view

The physical view of a software system is meant to represent the items that comprise it at runtime. With virtual being the new physical, and cloud being the new virtual, I’ll simplify this view as the components that own their own memory space. If you are describing the physical view for an application in production, you might draw a single virtual machine, but that wouldn’t be very descriptive for small applications. A better approach would be to draw the VM as a container and illustrate the different processes that might run on that VM and their dependency on one another. For the purposes of the DevOps environment that we are describing, please consider the following figure.



Figure: The physical view of a .Net DevOps Environment.

The physical view of our DevOps architecture shows the products that must be online and connected with one another in order to enable our DevOps environment. As you add products and other tools, this view of the architecture will grow.

This is a high-level physical view, as the three environments that we see depicted in the local view are just represented by an Azure icon in this view. It would be appropriate to split that out if we wanted to specify different regions for our environments. At the highest level, this is our physical view.

#### Development view

The development view, within your 4+1 architecture, depicts the structure of a developer’s workstation and surrounding resources in order to implement the system described by the architecture.



Figure: The development view for the DevOps environment described in this book

In order for you to develop your own world-class DevOps environment for your team, you will need the public Azure DevOps Services project provided with this book, the Git repository for the sample application, and this book along with access to the other books and text referenced in footnotes throughout this text. This book is not meant to stand alone. It is a guide through the complete .Net DevOps implementation provided with the book and delivered via Git and a public Azure DevOps Services project.

This book is not meant to stand alone. It is a guide through the complete .Net DevOps implementation provided with the book and delivered via Git and a public Azure DevOps Services project

Now that we’ve reviewed the four views of the architecture for our DevOps environment, let’s look at some scenarios that will be supported by this architecture.

#### Scenarios

The scenarios included with a 4+1 architecture are meant to illustrate the capabilities of the architecture. In the simplest form, scenarios can be use cases or even a list of capabilities the architecture provides. For complex scenarios, it can be useful to illustrate through drawing how data would flow through the system as someone used a capability of the system. For the purposes of our DevOps environment, being the system under design, a list of capabilities will suffice.

* A team member can see which features are in varying states of design by glancing at the project board
* A developer can open a new feature branch from a feature work item that is in development
* A developer can run a private build locally, without outside dependency, to validate readiness for a commit/push
* A developer can execute unit tests and integration tests locally to validate changes before pushing code to the team’s Git repository
* A developer can see newly pushed code build in a continuous integration build and know that the new changes worked well with changes from other teammates
* A developer receives notification of pass or fail of the full body of automated full-system tests that run in a fully deployed environment
* Any team member can access versioned release candidate packages for any application components of any successful build
* A developer can submit a pull request in order to have a teammate inspect the set of changes linked to the work item of the branch
* A pull request inspector can see the successful CI build, deployment, and full-system test run along with static analysis results and test code coverage metrics while executing the pull request inspection checklist
* A stakeholder can request on-demand for a new build to be deployed automatically to an environment and see it deployed quickly, database and all
* A team member can query the centralized logs from any environment in order to diagnose issues reported via configured alarms

Every team’s DevOps environment should include these base capabilities. Many teams will want even more capabilities. This book will enable you to design and implement a DevOps environment with the above capabilities. At this, the author hopes you are excited! If your team’s working environment contained all these capabilities enumerated, would you produce faster or slower for your customers? Would you produce better quality or more bugs with the above capabilities? Certainly, any team would be better off with these capabilities.

### DevOps methodology

Before we begin discussing how to implement the architecture of a DevOps environment, let’s review the state of the DevOps methodology in the industry. In his popular book, [The Phoenix Project: A Novel About IT, DevOps, and Helping ...](https://www.amazon.com/Phoenix-Project-DevOps-Helping-Business-ebook/dp/B00AZRBLHO)[[6]](#footnote-6), Gene Kim enumerates three guiding principles of DevOps. He names them “The Three Ways”. They are:

1. The First Way: Systems Thinking
2. The Second Way: Amplify Feedback Loops
3. The Third Way: Culture of Continual Experimentation and Learning

These are quite abstract but look out for them as we implement our DevOps scenarios specified above. In our industry lies much confusion about DevOps. Having only been named as such in 2010, DevOps has been commercialized and marketed. You’ll see job openings for “DevOps Engineer”. This is akin to the “Senior Agile Engineer” job postings around the 2005-time frame. DevOps is a way of thinking. It is a mindset. There are practices that go very well with the DevOps way of thinking, just like test-driven development goes very well with the Agile way of thinking. Let’s illustrate the Three Ways of DevOps briefly.

#### The first way: systems thinking

The DevOps methodology is based on the principles known as The Three Ways. The First Way is Systems Thinking.



Figure: The First Way is Systems Thinking

There is a lot of thought packed into this first principle of DevOps. It encompasses the ability to create a smooth, predictable flow of working software from the imagination of the developers to the active use of the customer. In our world, regardless of job description or job title, if you are involved in building or changing the software, you are on the side of Dev. If you are someone who uses or consumes or depends on the software, then you are on the side of Ops. Other definitions of DevOps that don’t include the user are at great risk.

#### The second way: amplify feedback loops

In this principle, we create an environment where those using our software – those operating their business or departments with our software – provide continual feedback to those operating, developing and changing the software.



Figure: Those operating their departments using our software provide continual feedback to those changing the software.

We can put ourselves in the right DevOps mindset by translating the keywords as follows:

* Dev: Includes anyone who works to support, build, change, improve the software or system
* Ops: Includes anyone who relies on the software to operate their business or department

If our company has a department known as IT Operations, or Support, or Data Center Operations, it’s important not to confuse these groups as our customer. They don’t use the software. They are merely part of our development capability. The capability to deliver valuable software to our customer so that our customer can operate the software in order to experience its value.

#### The third way: culture of continual experimentation and learning

If the first two Ways were about completing a software release cycle in a streamlined and effective way, The Third Way is about making that cycle faster.



Figure: The Third Way introduces smaller cycles within the cycle

Companies such as Netflix showed us that software can be released not only daily but many times per day with no downtime and no defects (or close to that ideal). The Third Way causes us to think with that end in mind, solving any challenges that would prevent us from this ability. Even if the customer doesn’t want software releases at that cadence, this way of thinking causes us to gain this capability so that we are ready on a moment’s notice to release the software as it stands, always stable, always working, always bug-free. This way of thinking also encourages us to stop thinking about software releases as a big ceremony. We will see in the coming chapters how to equip our teams with the ability to release changes big and small. We will see that the same process needed for small changes is effective for large changes when every small change has made a trip down the DevOps pipeline. Now that we have covered the architecture and the thinking of DevOps, let’s see how to get started.

## How to get started

The example application, along with the Azure DevOps Services configuration is available online as a public project.

<https://dev.azure.com/clearmeasurelabs/Onion-DevOps-Architecture>

Keep the sample application and the Azure DevOps project handy as you move through this book. This working sample serves to demonstrate all the capabilities working together. No sample application will be sufficient to illustrate every scenario in the development world, but for the purposes of a DevOps environment, we have chosen the most common application type at the moment. Before we review the sample application, let’s map common application components to their runtime components.

### Application runtime architectures

You are building software with Visual Studio. Regardless of the libraries or frameworks you might choose, you have a finite set of runtime architectures to choose from. Popular today is a web architecture which consists of a ASP.NET web application and a SQL Server database. Or you might have a desktop WPF application communicating to ASP.NET Web API services that then use a SQL Server database. If you have an iPhone app, that might connect to your Web API services. Regardless of the combination of runtimes you take advantage of, the Microsoft platform has a finite set of choices, and the Azure cloud has a handful of ways to run each that must be deployed into the cloud.

|  |  |
| --- | --- |
| Application architecture | Production Runtime |
| ASP.NET MVC | IIS/Azure App Service, etc. |
| ASP.NET Web Api | IIS/Azure App Service, etc. |
| SignalR | IIS/Azure App Service, etc. |
| Razor Pages | IIS/Azure App Service, etc. |
| SPA(Razor Components, Angular, React) | IIS/Azure App Service, etc. |
| Windows Service | VM, Webjob, Azure Function, etc. |
| Scheduled EXE | VM, Webjob, Azure Function, etc. |
| SQL Database | Azure SQL, Cosmos DB |

Table: A small illustration of Azure runtimes covering the breadth of application architectures.

While this table is nowhere near being complete, we can see that through web applications, offline jobs, and a relational database, we cover a high percentage of applications out in the wild. WPF, Winforms, and native iOS and Android applications are also supported by a small number of options. With each of these application types, we can choose a full range of runtime options from Infrastructure as a Service (IaaS) to Platform as a Service (PaaS).

The architectural point to consider when designing a DevOps capability is to realize that while implementing The First Way, we need **not** support a unique configuration for every team or application. Once we understand how to deliver a web application of some form with a SQL database out to Azure, how many of our applications are now covered? Most. I would venture to guess for each of you, dear readers, that a high number of your applications use those architectural components. We then add a capability for off-line jobs such as Windows Services and Scheduled Tasks, and we cover a good part of the gap. Once we have these application types covered, you will see how much smaller of a leap it is to then cover your native mobile apps, and Windows desktop apps as well.

### The necessary tools

In order to set up a professional DevOps environment targeting Azure, you’ll need to have a few key tools to get started.

* An Azure subscription.
* Visual Studio (2019 or VSCode)
* An Azure DevOps Services organization account

These tools are just the starting point, and throughout this book, we’ll integrate more tools, libraries, frameworks from Microsoft, other vendors, as well as open-source repositories. Remember, DevOps is about a way of thinking that leads to an outcome of shorter lead times, shorter cycle times, and fewer disruptions. Throughout this book, we’ll put all these pieces together one-by-one.

If you are just getting started with Azure or Azure DevOps Services, don’t skip Chapter 2. It will quickly introduce some basic capabilities in an interactive way without requiring you to write any scripting. But don’t stop there. The steps shown in Chapter 2 are only to introduce first-time users of these tools. These techniques are not meant for long-term maintainability. For the professional way to set up your DevOps Environment, move to Chapter 3 and beyond where we will go through each area in detail.

### The sample application

While samples tend to be too simplistic or unnecessarily complex, the book uses an ASP.NET Core web application configured with a SQL Server database, using Entity Framework Core for an object-relational mapper (ORM). This application is properly factored into logical layers that control access to key dependencies. The application only does one thing, which is expose a Web API that retrieves Expense Report records. Our core object model has exactly one entity, ExpenseReport, and our SQL Server database schema has exactly one table, ExpenseReport. Rather than clutter this book with general coding practices, this sample provides some structural guidance with enough surface area to demonstrate all the necessary DevOps techniques. It does not see to illuminate coding patterns or frameworks.

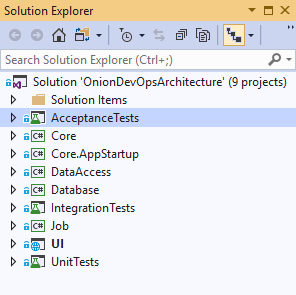


Figure: Solution structure of the sample application

The Visual Studio solution does conform to the Onion Architecture[[7]](#footnote-7) dependency management model, by isolating key dependencies inside their own assembly and concentrating application and business logic in a dependency-free assembly called “Core”.

* AcceptanceTests: The full system acceptance tests reside in this project. They run as NUnit tests configured with Selenium and drive the Chrome browser to execute tests against a fully deployed instance of the application.
* Core: This project has no Nuget package dependencies as well as no project dependencies. It is best implemented as a .Net Standard library, and it should contain plain C# objects. The value of this library is that any code it contains is verified to be portable to any application type given that the assembly produced will have no other dependencies that the base class library and C# language features.
* Core.AppStartup: This project exists to bootstrap the application, initiate the Inversion of Control (IoC) container and instantiate any global resources or cache’s that may be needed by the application. Some developers put this logic in the UI project because it is the startup point for the application, but since IoC has nothing to do with serving web pages, we have factored it out to its own very small project.
* DataAccess: This project maintains the responsibility of configuring and using the Entity Framework Core dependency. It contains logic to map our class to the SQL Server schema. It also contains any logic that requires using the EFCore APIs. No code outside of this project knows that the EFCore package exists.
* Database: The database project contains our SQL Server schema migration tooling and migration scripts necessary for incremental, automated changes to the database schema and non-user data.
* IntegrationTests: This project houses our L1 tests, or integration tests. This is described more in chapter 7.
* Job: This project houses a normal back end job that runs on a scheduled interval.
* UI: This project is an ASP.NET Core project and serves our ValuesController, returning all ExpenseReport(s). It makes use of the capabilities of the Core project.
* UnitTests: This project houses the L0 tests, or unit tests for our code.

This application is about as simple as it gets. The Visual Studio solution, however, is factored in a manner that would be suitable for a larger application or service. The hope is that this sample may be a starting point for your own applications, and it would be unrealistic to provide a sample that contained only a single project with no logical separation or dependency isolation.

## About the book

Using the model described above for a complete DevOps environment, each of the chapters in this book highlight in detail how to think about each area. The relevant principles will be covered first along with how to implement that part of the environment. While some specifics are highlighted in the chapter, this text makes heavy references to other books, articles, and Microsoft documentation. Rather than duplicate other works and documentation, which will be updated more rapidly than this book, footnotes are used to direct you to the right resource. In addition, use the sample application and public Azure DevOps Services (sometimes abbreviated hereafter as “AzDO”) to follow along while digging in as deep as you like. The number of integrations and settings required in order to establish a complete DevOps environment can be daunting. That’s why this book needs to be published. While there is no practical way to publish every setting and script file in the text, you may use the accompanying video-recorded walkthrough of the public AzDO project in order to gain a detailed understanding of all the settings that were changed from their defaults.

Until you have a complete DevOps environment, go “by the book”. Once everything is online and functioning, feel free to customize, change, and improve.

Now that you understand the challenge our industry is facing and the model for DevOps that will be implemented in this book, let’s review what will be covered as you read through the chapters.

Chapter 2 is for those new to Azure DevOps and to Azure in general. It will cover how to set up the basics so that you can follow along with the rest of the book. If you are already a user of AzDO and Azure, feel free to quickly skim the chapter and move on.

Chapter 3 moves beyond the quick starts and online tutorials and describes the professional grade development environment and the tools you should be using.

Chapter 4 dives into the first phases of work in our DevOps process, which is tracking work. You learn how to customize your own project board so that all work is visible.

Chapter 5 teaches how to track code using Azure Repos. More importantly, it teaches the natural rules for segmenting applications into Git repositories in a fashion suitable for the beginning of an automated DevOps pipeline.

Chapter 6 builds the code. The chapter covers the different types of builds and when to use each. After covering the steps that each type of build should contain, the reader is taken through the configuration of the continuous integration build in Azure Pipelines.

Chapter 7 Is the quality control chapter, illustrating how to think about code validation in your DevOps environment. This chapter relates relevant quality research along with rules of thumb for implementing the three required defect removal methods necessary in any DevOps environment.

Chapter 8 creates a versioned, deployable release candidate. This chapter shows how to decide the boundaries of application packaging, how many packages you should have, and how to create them in a deployable format and store them in Azure Artifacts.

Chapter 9 provisions, configures environments, and deploys our release candidates across the environments in Azure. This chapter covers the three distinct types of environments, the different among them, and how to dynamically create each using PowerShell, Azure Resource Manager (ARM) templates, and Azure Pipelines. It also takes the reader through the deployment steps needed for the three environment types and the configuration settings in Azure Pipelines that allow for total control of when and how software is released.

Chapter 10 rounds out the book by covering how to monitor and support software that has been deployed to a production environment. This chapter shows how to implement and centralize logs and other diagnostics so that they are available for analysis both for proactive alarms using Azure Monitor as well as for on-demand investigation.

## Wrap up

This is the beginning of a new era for your software team. Use the methods and examples in this book to constantly accelerate your team’s cycle time. Analyze any part of the process that causes delays or bottlenecks and squeeze those problems out. Now let’s get on with the book!

# Bibliography

Goldratt, E. M. (1990). *Theory of Constraints.* North River Press. Retrieved from https://www.amazon.com/Theory-Constraints-Eliyahu-M-Goldratt-ebook/dp/B00L7XYW2Q

Goldratt, E. M. (2014). *The Goal: A Process of Ongoing Improvement, 30th Anniversary Edition.* North River Press. Retrieved from https://www.amazon.com/Goal-Process-Ongoing-Improvement/dp/0884271951

Guckenheimer, S. (2018, 9 24). Sam Guckenheimer on Testing, Data Collection, and the State of DevOps Report - Episode 003. (J. Palermo, Interviewer) Retrieved from http://azuredevopspodcast.clear-measure.com/sam-guckenheimer-on-testing-data-collection-and-the-state-of-devops-report-episode-003

Kim, G., Behr, K., & Spafford, G. (2013). *The Phoenix Project: A Novel About IT, DevOps, and Helping Your Business Win*. Retrieved 2 18, 2019, from https://amazon.com/phoenix-project-devops-helping-business/dp/0988262592

Kruchten, P. (n.d.). Retrieved from Architectural Blueprints—The “4+1” View Model of Software Architecture: https://www.cs.ubc.ca/~gregor/teaching/papers/4+1view-architecture.pdf

Palermo, J. (n.d.). *The Onion Architecture*. Retrieved 3 21, 2019, from http://jeffreypalermo.com/blog/the-onion-architecture-part-1/

Wikimedia Commons. (2016). *File:Devops-toolchain.svg*. Retrieved from https://commons.wikimedia.org/wiki/File:Devops-toolchain.svg

1. (Kruchten) [↑](#footnote-ref-1)
2. (Wikimedia Commons, 2016) [↑](#footnote-ref-2)
3. (Guckenheimer, 2018) [↑](#footnote-ref-3)
4. (Goldratt, The Goal: A Process of Ongoing Improvement, 30th Anniversary Edition, 2014) [↑](#footnote-ref-4)
5. (Goldratt, Theory of Constraints, 1990) [↑](#footnote-ref-5)
6. (Kim, Behr, & Spafford, 2013) [↑](#footnote-ref-6)
7. (Palermo, n.d.) [↑](#footnote-ref-7)