Hi, welcome to the Introduction to Azure Bicep course on Udemy. I’m Dylan Budgen, I’m a software and DevOps engineer with over 5 year’s experience in industry, and I use Bicep everyday to deploy infrastructure as code. This course is the most thorough on Udemy and you will go from beginner all the way to expert in just one course.

Hi, I’m Dylan Budgen

Write up topics like deploying function apps on my blog

Release YouTube videos of the function app specific

TODO

* Functions video needs reviewing and cutting for the gaps in my thinking

Course Title

Description

Learning objectives

Preview image

Preview video

**Video Introduction**

* Hi, welcome to the Introduction to Azure Bicep course!
* Azure Bicep is a brand new powerful infrastructure-as-code tool for deploying and managing Azure resources. This new approach replaces difficult and impossible to read JSON ARM templates with an easy to use and streamlined language. With Bicep we are able to create very simple deployments, all the way to complicated large scale projects.
* This course is the most through on Udemy with real world examples for you to become an expert.

Introduction – what is bicep?

List of things they will learn

Fast forward of me writing a complicated bicep file

Pipeline deploying (validating then deploying)

\*\* the most through Udemy course

TODO

Watch existing videos on YouTube for recording ideas

Make face bottom corner on function app video and managing dependencies,

1. Introduction – chose best clips, add
2. Introduction videos – move to left, add title to video intro and add titles
3. Lesson 1
4. ~~Lesson 2 – cli on windows – rerecord~~
5. ~~Lesson 2 – powershell on windows – rerecord~~
6. ~~Lesson 2 – vs code – rerecord~~
7. ~~Some of the new videos need top cropped~~
8. ~~Lesson 2 - Deployment scopes – deploying an RG~~
9. ~~Some of the new lessons need the video cropping – these also have saturation issues~~
10. ~~Lesson 9 – pipeline validation -rerecord ending~~
11. ~~Lesson 3 – rerecord looking at camera~~
12. ~~Lesson 2~~
13. ~~Lesson 11~~
14. ~~Lesson 10~~
15. ~~Lesson 4 - powershell deployments~~
16. ~~Lesson 4 - What if deployments~~
17. ~~Lesson 4 – azure pipelines~~
18. ~~Lesson 13 – enabling preview features -maybe rerecord~~
19. Update github repo lessons
20. Make repo public
21. All videos which use visiualiser need video moving (function app, dependencies, storate key)
22. ~~Increase audio on all videos~~

After completion

1. Update CV
2. Update website
3. Update linkedin

Mac

deployment steps

az login

az deployment group create --subscription udemy-courses --resource-group bicep-course --name deployment --template-file main.bicep

**--mode**

Deploying resource groups, deployment scopes Where should I show them deploying resource groups?

Dependencies – implicit, explicit, Visualize dependencies

**Lesson 1: Introduction to Azure Bicep**

**Lesson 2: Setting up the environment**

**Lesson 3: Creating your first Azure Bicep template**

**Lesson 4: Deploying resources with Azure Bicep**

**Lesson 5: Variables, parameters and outputs**

**Lesson 6: Bicep functions**

**Lesson 7: Working with modules in Azure Bicep**

**Lesson 8: Advanced Bicep concepts (loops, conditional deployments, existing resources)**

**Lesson 9: Testing and validating Azure Bicep templates**

**Lesson 10: Troubleshooting Azure Bicep deployments**

**Lesson 11: Bicep best practices**

**Lesson 12: Real-world Project – deploying a function app with logging**

**Lesson 13: Preview features**

**Lesson 14: Review and Next Steps**

**Lesson 1: Introduction to Azure Bicep**

**Introduction to Infrastructure as Code (IaC)**

[Begin with a little excitement] "Hello and welcome, everyone! In our very first lesson, we are going to learn about Infrastructure as Code.

Infrastructure as Code is a key practice in the field of DevOps and involves the process of managing and provisioning your computing infrastructure through script files, rather than manual processes.

In the not-so-distant past, setting up infrastructure involved a lot of manual work. System admins had to physically install servers, configure networking equipment, and manually set up environments. It was time-consuming, error-prone, and difficult to replicate consistently.

But then, IaC came along. With IaC, we can automate all of those manual processes. We can write code to define our infrastructure, and then use that code to automatically set up and configure our servers, databases, networks, and more.

[Talk about the benefits] The beauty of IaC is that it's predictable, reusable, and efficient. It reduces the potential for human error, enables consistent deployments across different environments, and significantly accelerates the process of setting up infrastructure.

Benefits

Consistency in deployments

IaC removes the risk of “snowflake” configurations.

Reduced human error in deployments

With no humans involved in our deployments there is far less risk of errors

Faster deployments

Quicker deployments also means potentially reduced costs. I like to use the example of Virtual Machines running in a development environment. To manually recreate a VM would take many hours, but with IaC, it could take minutes. You could potentially tear down all your development environments when not in use and rebuild on demand, saving you money.

Source control

Since we are now using code to control our infrastructure, it can be subject to our source control procedures. This means any change can require approval and knowing the deployment configuration is far simpler.

**Introduction to Azure Bicep and ARM templates**

Before Bicep came along, Azure offered something called Azure Resource Manager templates, or ARM templates.

ARM templates are JSON files with a complicated looking structure used to define your infrastrcure deployments. These are declarative files so you define the end state of your infrastructure, and the Azure Resource Managers handles everything else. These templates are also indopoment, meaning we can deploy them as many times as we like, and the end result will be the same.

Microsoft realised that whilst ARM templates are incredibly powerful, the usability and readability are quite poor. So they introduced Azure Bicep.

Azure Bicep is a domain-specific language, or DSL, for deploying Azure resources declaratively. In simpler terms, it's a language designed specifically for defining and managing your Azure resources using code.

Bicep's syntax is cleaner and more straightforward than JSON, making it easier to read and write. Secondly, it's fully integrated with Azure tooling. This means you can use it with the Azure CLI, Azure PowerShell, and the Azure portal. It's also supported in your favorite IDEs, like Visual Studio Code, with the Bicep extension. Finally, Azure Bicep is a transparent abstraction over ARM templates. This means whatever you can do with ARM templates, you can do with Bicep.

Update to date compared to terraform

One of the key advantages of Azure Bicep is its clean and simplified syntax. Compared to the sometimes verbose and complex JSON syntax of ARM Templates, Bicep offers a more intuitive language that is easier to read, write, and maintain. This simplicity translates into increased productivity and reduced chances of errors.

The connection between bicep and ARM

Define the connection] So, what exactly is the relationship between Azure Bicep and ARM Templates? In simple terms, Bicep is a language that compiles down to ARM Templates. You write your infrastructure code using Bicep's user-friendly syntax, and behind the scenes, that code gets transformed into an ARM Template, which Azure can understand and deploy.

[

Verry current and always being updated

Host (On-Screen): First and foremost, Azure Bicep offers a simplified syntax and structure. Its syntax resembles JSON, making it easy to read and write. With Bicep, you can express your infrastructure requirements concisely and clearly, reducing the chances of errors and improving the overall development experience.

[Explain with an analogy] You can think of it a bit like how a high-level programming language, such as Python, gets compiled into machine code that a computer can understand and execute. With Azure Bicep, you're writing your infrastructure code in a more human-friendly language, but what gets deployed to Azure is still an ARM Template.

Doesn’t require state

It also leverages the power of ARM and doesn't require a state management solution like Terraform.

!"

[End of Video]

[Title Slide: Introducing Azure Bicep]

Azure Bicep is a brand new

Host (On-Screen): Azure Bicep is a domain-specific language (DSL) designed specifically for Azure deployments. It provides a simplified and streamlined approach to provisioning and managing Azure resources, making infrastructure-as-code more accessible and efficient.

[Slide 1: Benefits]

Host (On-Screen): So, why should you consider using Azure Bicep? Well, let's take a look at some of its key benefits.

[Slide 2: Simplicity]

Host (On-Screen): First and foremost, Azure Bicep offers a simplified syntax and structure. Its syntax resembles JSON, making it easy to read and write. With Bicep, you can express your infrastructure requirements concisely and clearly, reducing the chances of errors and improving the overall development experience.

[Slide 3: Familiarity]

Host (On-Screen): Another advantage of Azure Bicep is its familiarity. If you've worked with Azure Resource Manager (ARM) templates before, you'll find Bicep quite familiar. Bicep is built on top of ARM templates, providing a higher-level abstraction that simplifies the authoring process while still leveraging the power of ARM underneath.

[Slide 4: Integration]

Host (On-Screen): Azure Bicep integrates seamlessly with Azure. It leverages Azure Resource Manager (ARM) to deploy and manage resources, allowing you to take full advantage of Azure's extensive capabilities. Bicep makes it easy to define and deploy complex Azure infrastructure with just a few lines of code.

[Slide 5: Maintainability]

Host (On-Screen): With Azure Bicep, maintaining your infrastructure becomes more manageable. Bicep supports modularity and reusability through modules, allowing you to organize your codebase efficiently. Changes and updates can be made easily, and you can track the state of your infrastructure using version control systems.

[Slide 6: Collaboration]

Host (On-Screen): Collaboration is crucial in any development process. Azure Bicep promotes collaboration by enabling teams to work together on infrastructure-as-code projects. You can split your code into multiple Bicep files, assign specific sections to different team members, and merge their changes seamlessly.

[Slide 7: Ecosystem]

Host (On-Screen): Azure Bicep benefits from being part of the broader Azure ecosystem. It integrates with Azure DevOps, Azure CLI, and Azure PowerShell, providing a cohesive workflow for your Azure deployments. Additionally, Bicep leverages existing ARM templates, enabling you to reuse and leverage the vast collection of ARM templates available.

Host (On-Screen): In conclusion, Azure Bicep offers simplicity, familiarity, seamless integration with Azure, improved maintainability, enhanced collaboration, and an extensive ecosystem—all aimed at making infrastructure-as-code on Azure more efficient and accessible.

Host (On-Screen): Throughout this course, we'll dive deep into Azure Bicep, exploring its features, best practices, and practical examples to help you become proficient in deploying and managing Azure resources using Bicep.

Host (On-Screen): That wraps up our introduction to Azure Bicep. Get ready for an exciting journey into the world of Azure infrastructure-as-code. Happy learning!

* 



Windows

<https://learn.microsoft.com/en-us/cli/azure/install-azure-cli-windows?tabs=azure-cli>

**Lesson 2: Setting up our environment**

Hello everyone, and welcome! In this lesson, we're preparing for our journey into Bicep by setting up our development environment.

We'll be walking through the installation process for all the necessary tools across both macOS and Windows platforms. Whether you're a Mac user or a Windows enthusiast, we've got you covered.

First, we'll start with the installation of Bicep itself, the core tool we'll be using throughout this course. Then, we'll guide you through the setup of Visual Studio Code, one of the most popular code editors out there, to tailor it specifically for Bicep development.

As part of our VS Code setup, we'll install use useful extensions. These tools will augment our coding experience with features like syntax highlighting, autocompletion, and linting - turning VS Code into a powerful, Bicep-focused IDE.

By the end of this lesson, you'll have a fully equipped, ready-to-go Bicep development environment. This will set the stage for all the coding and learning we'll be doing in the lessons to come.

Let's dive in!

**Install-Module -Name Az -Repository PSGallery -Force**

Mac

brew install azure-cli

az bicep install

windows

install

* Setting up the environment for Bicep
* VS code, bicep extension

**Lesson 3: Creating your first Azure Bicep template**

Hello everyone, and welcome! Today is a significant day in our journey with Bicep, as we will be creating our first Bicep file. This is where we start translating theory into practice, laying a solid foundation for your future Bicep projects.

In this lesson, we'll take it slow and simple. Our task will be to create a straightforward template for an Azure Storage Account. It may seem basic, but even the simplest script requires a proper understanding of the Bicep language and Azure resource properties.

One of the most crucial skills we'll use and develop in this lesson is navigating and utilizing the official Bicep documentation. This is your go-to source of information for Bicep's syntax, resource types, and API versions. Learning to effectively leverage this documentation is vital in mastering Bicep and overcoming any coding challenges that may come your way.

By the end of this lesson, you'll have written your first Bicep file and discovered how to use official documentation to find Bicep APIs. Remember, the journey of a thousand miles begins with a single step, and today we take that step. Let's get started!

* Create your first Bicep file
* Converting bicep file to ARM template

We are going dive straight in and create our first Bicep deployment template. So we need to create a new file with the bicep extension, which I’m I’m going to call main.bicep.

* Declare resource with resource
* Type name of the resource – this is a reference for the resource in the bicep file
* Autocomplete the resouece type, choose the API version
* Use autocomplete for the required properties
* Type the name, this is the name of the resource
* Location – hardcode to wesreurope

**Lesson 4: Deploying resources with Azure Bicep**

Hello everyone, welcome back! Today, we're going to roll up our sleeves and get practical with deploying resources in Azure using Bicep. This is a critical step in your journey to becoming proficient with Infrastructure as Code.

We'll walk you through multiple paths for deploying your resources. Each of these methods serves its purpose and understanding when and how to use them will significantly enhance your efficiency and flexibility.

Next, we're going to transition into the command line interface (CLI), an essential tool for automation and scripting. CLI offers speed and direct control over your deployments, especially when you're handling multiple resources.

Following the CLI, we'll move onto PowerShell. As a task-based scripting language built on .NET, PowerShell offers a robust platform for managing your resources in Azure, especially if you come from a Windows-centric environment.

And lastly, we'll delve into Azure DevOps, Microsoft's end-to-end DevOps toolchain. We'll demonstrate how you can leverage its built-in deployment pipelines to manage and automate your Bicep deployments, making your workflow more streamlined and efficient.

But that's not all! At the end of the lesson, we're going to learn how to deploy resource groups. This is a critical skill as resource groups serve as logical containers for resources deployed on Azure.

By the end of this session, you'll have a strong understanding of various deployment techniques and the contexts in which they are most effective. Let's dive in and master the art of deploying resources in Azure with Bicep!

* Overview of deployment options Azure CLI, PowerShell, Azure DevOps, etc.)
* Show a deployment in the portal
* Deployment of Bicep files with Azure CLI
* Deployment of Bicep files with PowerShell
* Integrating Bicep deployments of Bicep files with Azure DevOps Pipelines
  + <https://dev.azure.com/dylanbudgen/udemy-courses/_build/results?buildId=2&view=logs&j=12f1170f-54f2-53f3-20dd-22fc7dff55f9>
* What are deployment modes? .
  + Validation, Incremental, Complete

New-AzResourceGroupDeployment -Name deployment -ResourceGroupName bicep-course -TemplateFile main.bicep

az deployment group create \

--subscription udemy-courses \

--resource-group bicep-course \

--name deployment \

--template-file main.bicep \

az deployment group create \

--subscription udemy-courses \

--resource-group bicep-course \

--name deployment \

--mode Complete \

--template-file main.bicep

**Lesson 5: Variables, parameters and outputs**

Hello and welcome back. Today's lesson is all about the foundational building blocks of Bicep scripting: parameters, outputs, variables, and data types.

Each of these elements plays a vital role in creating readable, efficient, and reusable Bicep files. We'll break each concept down to its core and give you a thorough understanding of their purposes and best practices.

We begin with parameters, the user-defined inputs that give our scripts flexibility and reusability. We'll see how they can help tailor deployments to different scenarios without changing the script itself.

Next, we'll talk about outputs, the way our Bicep files communicate the results of their operations back to us. This feature is key for tracking deployments and chaining together multiple scripts.

Moving on, we'll discuss variables, our go-to for storing and managing information within our scripts. Understanding variables is essential for keeping our scripts clean and our deployments consistent.

Lastly, we'll cover data types, the categories that determine what kind of data our parameters, outputs, and variables can hold. A solid grasp of data types ensures we use each of these elements effectively and avoid common mistakes.

By the end of this lesson, you'll be well-equipped with these core Bicep concepts, taking your scripting skills to a new level. Let's get started!

* Using variables in Bicep
* Using parameters in Bicep
  + Lower camel case
* Using outputs in Bicep
* Data types – mention naming conversions – put it in the video as a pop up!

Removing hard coded values, and making our bicep templates more reusable

Variables, remove duplication and allow us to have more interesting and complicated templates

Outputs

Data types – we will explore with demos all of the different data types bicep offers and how we can use them in our template

**Lesson 6: Bicep functions**

Hello everyone, and welcome back! In this lesson, we're focusing on a powerful feature of Bicep: Functions.

Bicep functions are at the heart of creating dynamic, flexible scripts. They provide us with the ability to manipulate and process data, interact with Azure resources, and control the flow of our scripts, making them an essential tool in our Bicep arsenal.

Throughout this lesson, we'll dive into different types of functions, learn their syntax, and understand how and when to use them. We'll cover everything from simple, single-purpose functions to more complex ones that can drastically enhance the capabilities of your scripts.

As we explore, we'll also get hands-on with some practical examples. This will not only deepen your understanding of the subject but also illustrate how these functions come to life in real-world scenarios.

By the end of this lesson, you'll have a strong command over Bicep functions, giving you the ability to create more efficient, flexible, and powerful Bicep files.

So, let's dive in and learn how to make the most out of functions in Bicep!

* Introduction to Bicep functions
* Commonly used Bicep functions
  + Scope functions
  + Resource functions
  + Guid functions
  + Array functions
  + String functions
  + Numeric functions

**Lesson 7: Working with modules in Azure Bicep**

**Hi, welcome back. In this lesson we are going to learn about modules in Bicep.**

**Bicep allow you to organise your deployments with modules, which are reusable bicep files which can be deployed** from another Bicep file. With modules, you improve the readability of your Bicep files by encapsulating complex details of your deployment. You can also easily reuse modules for different deployments.

For example, if you often deploy the same combination of a web app and a SQL database, you can define them in a module. Then, instead of writing the same code again and again, you can simply call that module with the specific parameters each time you need to deploy that combination. This not only saves time but also reduces the chances of errors and inconsistencies in your code.

We may also want to modularise specific resources so writing your infrastructure becomes simpler. We can make a module for our storage account, setting all the properties on the account which we have decided are appropriate. Then anyone using the module simply injects any parameters, like a storage account name, and they don’t need to worry about any details of underlying infrastructure.

To share modules with other people in your organization, create a [template spec](https://learn.microsoft.com/en-us/azure/azure-resource-manager/bicep/template-specs), [public registry](https://github.com/Azure/bicep-registry-modules), or [private registry](https://learn.microsoft.com/en-us/azure/azure-resource-manager/bicep/private-module-registry). Template specs and modules in the registry are only available to users with the correct permissions.

You can even use ARM templates as a module in your bicep file.

* What are Bicep modules?
* How to create and use Bicep modules.
  + Storage account, create one manually, then change to module
  + Explain that it creates new deployment – show in the deployment
  + Explain the name different module and resource

**Lesson 8: Advanced Bicep concepts (loops, conditional deployments, existing resources)**

Hello everyone, and welcome back! Today we're taking a deep dive into some advanced Bicep concepts that will significantly enhance your Infrastructure as Code capabilities. We'll be tackling several important topics that build on your existing knowledge and add a layer of sophistication to your Bicep scripting.

Our first stop is exploring the parent/child relationships in resources using nesting. Here, we'll bring to life this concept by creating blob containers. Just as a container rests within a storage account, you'll see how resources relate to one another in a nested structure, setting up a clear hierarchy.

Next, we turn our attention to loops, a powerful tool that can help you avoid duplication and make your scripts more efficient. If you've ever needed to deploy multiple instances of the same resource type, loops will be your new best friend.

From there, we'll delve into how we can interact with existing resources in Bicep. It's not uncommon that our deployments will have to interface with resources that are already in place, and we'll navigate this scenario by creating role assignments.

Moving on, we'll tackle the concept of managing dependencies in Bicep. As your deployments grow more complex, understanding how Bicep determines the order of resource creation becomes essential. Here we'll differentiate between implicit and explicit dependencies.

We'll then explore the concept of conditional deployments. This technique gives you the power to deploy resources under specific conditions, perfect for tailoring resources to particular environments or requirements.

Finally, we'll examine the use of ternary operators in Bicep, a flexible tool that allows you to write more concise and conditional logic in your templates.

By the end of this lesson, you'll be equipped with a host of advanced Bicep techniques, ready to tackle more complex IaC tasks. So let's get started and elevate your Bicep proficiency to the next level.

Hi, welcome back. In this lesson, we are going to explore some advanced Bicep concepts to allow you to make professional and powerful bicep templates.

Firstly, we explore how we define parent/children relationships with resources with nesting, and to do that we add the option of container deployments to our storage account module.

Next we explore loops, and how we can use them to reduce duplication when deploying several resources of the same type.

Following that, we look at how we can interact with existing resources in Bicep. It is not uncommon that our deployments will have to have context of existing resources and we explore this topic by creating some role assignments.

We’ll also learn how to manage dependencies in Bicep. When we have more complicated deployments, we will need to manage how our bicep knows in what order to create resources. In this topic we explore the differences between implicit and explicit dependencies.

Next, we will explore how we can use conditional deployments. This allows use to optionally deploy resources under specific conditions. Perhaps you can only want to deploy specific resources to certain environments or under certain conditions.

Then, we explore how we can use powerful ternary operators in Bicep.

Finally, we'll take a closer look at ternary operators, exploring how to utilize them to create more intricate templates.

Finally, we explore ternary operators and how we can use them to create more interesting templates.

* Nested resources in Bicep
  + creation of single container in storage account module
* Loops in Bicep
  + Creating array of containers (follow up from parents)
* Explore different types of loops?
* Working with existing resources
  + role assignments (also show them with a different RG)
  + indexed loops
  + introduce with a single role assignment, then add loop for complex loops
* Managing dependencies in Bicep2 Lo
  + RBAC module being run after the storage accounts deployed
  + Modules vs resources – show this by creating a storage account without the module
  + Dependency management with resources and modules
  + Implicit vs explicit (explicit is discouraged)
* Conditional deployments in Bicep
  + audit storage account with Boolean param
* Ternary operators in Bicep
  + The conditional deployment will need a local variable for storage account names in the RBAC module – follow up from last module
* **Logical operators**

**Lesson 9: Testing and validating Azure Bicep templates**

Hello everyone, welcome back! Today's lesson is a critical one as we delve into the art of testing and validating your Bicep templates. Ensuring your code is error-free and aligns with best practices is key to maintaining robust, reliable deployments.

First, we'll demonstrate how to use the Azure CLI for validating your templates. This is a fantastic tool that enables you to catch errors early, without even leaving your terminal.

Next, we'll explore the power of linting in Visual Studio Code. This feature provides real-time feedback as you type, helping you spot potential issues and maintain good coding standards. It's like having an ever-vigilant pair of eyes that ensures you're writing clean, efficient code.

Then, we'll take our testing to the next level with validation in Azure Pipelines. This process automatically checks every change in your code before it's deployed, preventing problematic code from affecting your deployments. We'll walk you through setting up a validation pipeline and show you how it can save you from costly deployment errors.

Finally, we'll introduce you to the ARM Template Test Toolkit (ARM-TTK) library. This comprehensive toolkit goes beyond simple validation, providing you with a set of tests designed to ensure your templates adhere to the ARM best practices.

By the end of this lesson, you'll have a strong grasp of the different tools and techniques for testing and validating your Bicep templates. So, let's get started and ensure your Bicep code is of the highest quality!

* Validating Bicep using Azure CLI
  + Using linter in vs code
  + Az bicep build
    - Reports warnings
* Testing Bicep and ARM templates
  + Arm-ttk (doesn’t look very simple)
  + Installation and running tests
  + Mention API versions and why you would wantto skip tests
* Validating Bicep with Azure DevOps Pipelines
  + <https://dev.azure.com/dylanbudgen/udemy-courses/_build/results?buildId=2&view=logs&j=12f1170f-54f2-53f3-20dd-22fc7dff55f9>

**Lesson 10: Troubleshooting Azure Bicep deployments**

Hello everyone, welcome back! Today's lesson is all about navigating the waters of troubleshooting Azure Bicep deployments. As with any language or platform, encountering issues and bugs is part of the journey. However, it's not the errors we make but how we resolve them that truly shapes our learning and expertise.

In this session, we're going to arm you with the skills to identify, investigate, and resolve some of the most common issues that can occur during Bicep deployments. These skills are not just about fixing problems but also about understanding how Bicep works and why things might go wrong.

We'll start by introducing you to the common types of deployment issues you might encounter. These include syntax errors, incorrect resource configurations, and issues with dependencies, to name a few.

Next, we'll dive into the art of debugging. We'll walk you through how to interpret error messages, where to find detailed logs, and how to use tools like the Azure portal, Azure CLI, and Visual Studio Code to get to the root of a problem.

But we won't stop at just identifying the problems. For each type of common issue, we'll also explore potential solutions and best practices to avoid similar pitfalls in the future.

By the end of this lesson, you'll be well-equipped to handle any bumps on the road to mastering Azure Bicep. So, let's embrace the challenges and turn them into opportunities for learning. Let's get started!

* Debugging Bicep deployments and troubleshooting common issues
* Understanding deployment logs and error messages
  + Template validation (predeployment) - Storage account wrong length
  + Deployment failures – invalid value for minimumTlsVersion
  + Mention that nested resources sometimes don’t produce good results, look at deployment logs

**Lesson 11 – Bicep best practices**

Hello everyone, welcome back! In today's lesson, we're going to delve into the realm of Bicep best practices. When it comes to writing efficient, maintainable, and robust code, knowing the rules of the road is absolutely essential.

Bicep, like any other language, has a set of recommended practices. These practices are based on collective wisdom from countless developers, lessons learned from many projects, and a deep understanding of the language itself.

In this session, we'll explore these best practices in detail. We'll cover everything from organizing your code and naming conventions, to optimizing resource deployments and managing dependencies.

We'll also dive into the importance of modularizing your code, the art of writing reusable modules, and how to manage your parameters, variables, and outputs for maximum clarity and usability.

But it's not just about the code itself; it's also about the process. We'll discuss practices for testing and validating your Bicep files, strategies for troubleshooting, and methods for continuous integration and deployment.

These best practices aren't just arbitrary rules; they're proven strategies that will help you write code that's easier to understand, maintain, and debug. They'll also help you avoid common pitfalls and streamline your work process.

By the end of this lesson, you'll not just know how to write Bicep code, but you'll know how to write it well. So let's dive in and start crafting code that is truly a cut above the rest. Let's get started!

**Hi, welcome back. In this lesson we are going to cover some essential best practices in Bicep.**

<https://learn.microsoft.com/en-us/azure/azure-resource-manager/bicep/best-practices>

**Lesson 12: Real-world project – deploying a function app**

Hello everyone, welcome back. We've covered a lot of ground in our previous lessons, delving into a wide range of advanced Bicep concepts. Now, it's time to bring all that knowledge together and apply it to a real-world scenario.

In this final lesson, we're going to construct a Function App from the ground up, with an App Service Plan, Application Insights, and a Storage Account. This is a common configuration that you might find yourself needing to deploy regularly, so understanding it in depth can save you a lot of time and effort in the long run.

We’re going to combine everything we’ve already learned, such as modules, managing dependencies, loops, interact with existing resources, use conditional deployments and, of course, leverage those handy ternary operators.

On top of that, we'll also work on creating a modular structure for all the resources. Modularity is crucial for maintainability and reusability, and it can greatly improve the efficiency of your Infrastructure as Code practices.

By the end of this lesson, you'll not only have a comprehensive, practical understanding of how to deploy a Function App with Bicep, but also an enhanced ability to use advanced Bicep concepts to handle complex, real-world scenarios.

Let's dive in and bring together everything we've learned so far.

* Building a real-world infrastructure setup using Bicep
  + Deploy using pipeline with variables?
* Think of how I can use advanced topics
  + Type the compute module with the wrong type of dependencies values, then correct it
  + Also show them that storageAccount.name is the deployment name

**Lesson 13: Preview features**

Hi, welcome back. In this lesson, we're going to venture into the future of Bicep by exploring some of its preview features.

As an evolving language, Bicep is continuously being updated with new functionality. Some of these new features are released to the community for testing and feedback before they're fully integrated into the language.

In this lesson, we will explore some of these promising features, giving you a glimpse into the future capabilities of Bicep. Remember, while preview features can be exciting and offer new capabilities, they are still under development, and shouldn’t be used in a production environment.

By staying ahead of the curve and learning about these features early, you'll be ready to implement them as soon as they are officially released and deemed stable. This will not only expand your Bicep toolkit but also make you a more adaptive and future-ready Infrastructure as Code professional.

This topic will be updated as new preview features are released. So, let's get started and uncover the future of Azure Bicep.

* How to enable preview features
* Object types
* **extensibility**: Allows Bicep to use a provider model to deploy non-ARM resources. Currently, we only support a Kubernetes provider. See [Bicep extensibility Kubernetes provider](https://learn.microsoft.com/en-us/azure/azure-resource-manager/bicep/bicep-extensibility-kubernetes-provider).
* **paramsFiles**: Allows for the use of a Bicep-style parameters file with a terser syntax than the JSON equivalent parameters file. Currently, you also need a special build of Bicep to enable this feature, so is it inaccessible to most users. See [Parameters - first release](https://github.com/Azure/bicep/issues/9567).
* **sourceMapping**: Enables basic source mapping to map an error location returned in the ARM template layer back to the relevant location in the Bicep file.
* **resourceTypedParamsAndOutputs**: Enables the type for a parameter or output to be of type resource to make it easier to pass resource references between modules. This feature is only partially implemented. See [Simplifying resource referencing](https://github.com/azure/bicep/issues/2245).
* **symbolicNameCodegen**: Allows the ARM template layer to use a new schema to represent resources as an object dictionary rather than an array of objects. This feature improves the semantic equivalent of the Bicep and ARM templates, resulting in more reliable code generation. Enabling this feature has no effect on the Bicep layer's functionality.
* **userDefinedFunctions**: Allows you to define your own custom functions.
* **userDefinedTypes**: Allows you to define your own custom types for parameters. See [User-defined types in Bicep](https://aka.ms/bicepCustomTypes).

**Lesson 14: Review and Next Steps**

Review of the course

Next steps for learning more about Bicep and IaC