Dog Food Con 2018

By Andrew Hinkle



Slow clap to rounding applause. Well done, well done, [Dog Food Con 2018](https://dogfoodcon.com/). Every session I attended provided a wealth of information and in some cases immediate benefit to projects I'm currently working on. This is a rare event where the stars aligned in my favor.

When you attend these types of conferences you typically try to pick sessions to learn and grow technically and personally. You pick up sessions of up-and-coming trends to help prepare for what is coming next. I've typically come out of these conferences envious of what others are doing, ecstatic that such tech exists. Last year during the conference I was inspired to dip my virtual toes into learning the LUIS AI bot and have been hungry to learn more Azure features. This year's conference did not disappoint.

If you're not interested in reading my rant on adding what you will learn section for sessions, then feel free to [skip down to the session reviews below](#_Thursday_Keynote_–).

Contents

[Please identify what "you will learn" for each session 2](#_Toc527294341)

[Thursday Keynote – Grow, Build, and Cultivate Diversity: Using Inclusion as a Catalyst for Innovation in the Age of Disruption – by Nicole Jackson 4](#_Toc527294342)

[Advanced IaC with PowerShell and ARM Templates – by Vince Fabro 4](#_Toc527294343)

[Continuous Delivery: How GitHub deploys GitHub – by Christian Weber 5](#_Toc527294344)

[Introducing Domain-Driven Design – by Steve Smith 6](#_Toc527294345)

[Addition by Abstraction: A conversation about abstraction and dynamic programming – by Jerren Every 7](#_Toc527294346)

[From Zero to Serverless – by Chad Green 8](#_Toc527294347)

[Friday Keynote – Building Effective Enterprise Software Communities – by Christian Weber 10](#_Toc527294348)

[Baseball, Actors, and Bots – by David Hoerster 10](#_Toc527294349)

[Dipping your toe into Cloud Development with Azure Functions – by Brian T. Jackett 11](#_Toc527294350)

[Microsoft Teams – The Missing Manual – by Ricardo Wilkins 13](#_Toc527294351)

[Building an Ultra-Scalable API Using Azure Functions Without Too Much Worry – by Chad Green 14](#_Toc527294352)

[Conclusion 17](#_Toc527294353)

# Please identify what "you will learn" for each session

I overheard a few rumblings that some sessions felt more like marketing pitches. This perception is common when you attend a session that targets the problem you are having perfectly, but the presentation utilizes a product or technology not available to you. This can be frustrating, especially when you don't figure it out quickly enough to switch sessions. Imagine attending a session that spends half the time setting up a scenario and describes the high level architecture to solve the problem only to find out that the presenter's big reveal was to use an AWS service instead of the Azure service you were expecting. How dare they, this is a Microsoft conference after all! :P

To make the best of it I've asked the presenters questions. Why did you choose to work with that product over others? What were the other options? Why was the product you were expecting to hear about not used? Do you have any recommended resources to help me choose between products? Do the other products work similarly? If you see others not paying as close attention or look disconnected (not just because they look hungry), then ask the questions during a pause otherwise respectfully wait until the end of the session.

In the end all sessions are at their heart a marketing session. Their goal is to identify a problem and get you to buy into their solution to the problem. See what I did there? The solution may be a new concept and just as likely as a new shiny product, <cough>Azure</cough>.

Why did this happen, why do you feel so jaded? I pose that the session information did not describe the session well enough. Most conferences seem to follow this pattern. So how can we improve upon this? Let's start with the simple Who/What/When/Where/How.

* Who?
  + Who is the presenter? Covered in the session notes, check.
  + Who is the target audience? Covered in the session notes, check.
    - I like how Dog Food Con has categorized the sessions as SQL/BI, Front End/UX, AI/ML, Azure/Cloud, Infrastructure/Security, AppDev, Human Skills, DevOps, Microsoft Support, Blockchain, Dynamics, and O365.
* What is the goal of the session?
  + What is the problem we are trying to solve? Maybe in the session title/descriptions.
  + What tool/technique/architecture/whatever are we using to solve the problem? Maybe in the session title/descriptions.
  + What is the cost?
    - Free/Paid Plan/Consumption
    - Perhaps target company size is better? Individual/Small/Mid/Large/Enterprise/Global Dominance of all mere mortals.
    - This info would be nice, but I'm not convinced it should be included in the session notes. If I had the tool names before the session, I'd probably do enough research to know if the company already owned it or might reasonably consider.
* When is the presentation held? Covered in the session notes, check.
* Where is the presentation held? Covered in the session notes, check.
* How? This is the presentation. It takes you from not knowing to knowing.

The problem I think people are having is the title/descriptions are not always clear enough. I've seen sessions that are very explicit about the tool being used such as "Microsoft Teams – The Missing Manual" and the expectation in the description was to use it effectively. It was spot on and straight forward. We use Teams at work and we could definitely get some tips on using it better, sold.

On the other hand, another session I attended was "Baseball, Actors, and Bots". It was a great session that explained how a dashboard could be kept current with live information. What it neglected to include was "what" actor system was used, C# Akka.net, "what" was an actor system, event-driven system with a bounded context concept (DDDish), "what" bots were used, Luis bot, and "what" tool was used to keep the info live, SignalR. I have some interest in DDD and event handling, so I would like to understand how this actor system would be beneficial. Of course since I'm working on a LUIS bot, I would have been totally sold if that was mentioned specifically. I almost chose another session, but I'm so glad I attended it!

Dog Food Con added a Category to target the audience, but perhaps another sub-category "You will learn:" would help. Almost like your list of skills on a resume with the most prominently discussed/important skills listed first. You will learn: Teams. You will learn: C# Akka.net (an event driven actor system), Luis Bot, and SignalR.

Example:

* Baseball, Actors, and Bots
* Fri 9:50 am
* Polaris Room
* Category: AI/ML
* David Hoerster
* **You will learn: C# Akka.net (an event driven actor system), Luis Bot, and SignalR**
* Did you ever wonder how sports sites (like ESPN) have a dashboard of various games that update in near real-time Let's build a simple sports dashboard that will display near real-time updates based upon data being streamed in and processed by an actor system and interact with it using Bots.

With the "You will learn" section added, the session info feels more complete and clear. The nice part is the human skills can take advantage of the session info. You will learn: to step away from a computer and talk to a real person. Hmm… you can do that?

# Thursday Keynote – Grow, Build, and Cultivate Diversity: Using Inclusion as a Catalyst for Innovation in the Age of Disruption – by Nicole Jackson

You live in two worlds, personal and professional. You have your own background. You have your own opinions and perspectives. You are unique. You are open to learn from others and listen regardless of job title, race, gender, religion, and politics. You have experiences and lessons learned to share and so do they. You treat them with respect and as equals, blend ideas, contribute, encourage them to contribute, all with positive reinforcement and support, and as a result your team is ready to innovate.

# Advanced IaC with PowerShell and ARM Templates – by Vince Fabro

**You will learn: Advanced IaC with PowerShell and ARM Templates (nailed it!)**

Slideshows were not available at the writing of this review.

Managing your [Infrastructure as Code (IaC)](https://en.wikipedia.org/wiki/Infrastructure_as_Code) simplifies and automates the provisioning of resources such as VMs. Vince recommends using PowerShell (core for cross platform) and [Azure Resource Management (ARM Templates)](https://github.com/AppliedIS/azure-blueprint/wiki/ARM-Templating). If you don't use Azure he said you could also use [Ansible](https://www.ansible.com/) (recommended), [Terraform](https://www.terraform.io/), [Chef](https://www.chef.io/chef/) (not recommended), and [Puppet](https://puppet.com/solutions/infrastructure-as-code).

A basic ARM Template is a JSON file with the properties of schema, contentVersion, parameters, variables, resources, and outputs. Properties can contain functions such as: "sharedTemplateUrl": "[concat(parameters('storageContainerUri'), 'Shared/')]". This gives you the flexibility to build properties based on the other properties. Using parameter file transformations you can tokenize properties with "%token%", which is nice because your tokens may contain additional tokens and the tokens are processed until none are left. While many Azure resources provide a link to generate ARM templates, not all of them work without some tweaking.

At a high level the PowerShell script will follow these steps.

1. Create an Azure Storage account
2. Copy ARM parameters to storage
3. Create target resource group
4. Deploy to resource group
5. Delete ARM from storage

Vince warned us that PowerShell and ARM while powerful are not considered 1st class citizens.

* PowerShell has intellisense, ARM does not
* No compile time error, only runtime errors
* Error messages are not descriptive and typically very deceptive of the root cause

When deploying multiple resources, you'll want to switch to using PowerShell foreach –parallel. However, be aware that the ARM template must then switch to inline scripts (tasks) that don't know about your session, so you'll have to setup connections and other steps again.

Use [Blue-Green deployments](https://docs.cloudfoundry.org/devguide/deploy-apps/blue-green.html) to minimize downtime by having one public production environment and a second private production environment. Once deployments are finished and tested to the private production environment, change the router so the private environment is now public and live and the public environment is now private and idle.

Vince was an excellent presenter and evenly paced the session. He handled one "eager" listener very well when early in the session he questioned the difficulties associated with this type of automation. Halfway through the session once the advanced portion started, the difficulties and approaches made since. I'll definitely pass this info on to my colleagues as we start researching this type of automation.

# Continuous Delivery: How GitHub deploys GitHub – by Christian Weber

**You will learn: GitHub Projects overview, deployments using Hubot (GitHub only), and WebHooks**

[While I have a GitHub site where I post my tips and whitepapers](https://github.com/penblade), I'm no expert as most of my time is spent in VSTS. Christian's intro gave some basic highlights on how to use the tabs: Issues, Pull requests, and Projects. They sounded like close parallels to what I do in VSTS. [Issues](https://help.github.com/articles/about-issues/) track to do's, bugs, feature requests, and more, similar to VSTS epics, features, user stories, tasks, and bugs. [Projects](https://help.github.com/articles/about-project-boards/) are similar to VSTS projects allowing you to organize issues. [Pull Requests](https://help.github.com/articles/about-pull-requests/) seem to be similar to code reviews pending review before checking-in/merging, but for everything done since the pull request was made.

Christian highly recommends opening a Pull Request as soon as you start work on a request/project. This makes it easier to see all changes and let others contribute or make recommendations early in the process. Projects include [Webhooks](https://developer.github.com/webhooks/) that trigger off an event such as after successfully committing to a branch deploy to an environment. They test their branches in canary (test) environments as any good enterprise company should.

I was caught by surprise to hear how GitHub manages there branches and deployment. The master branch is always what is in production which makes since. However, they deploy to production services from their feature branch. If there is any problem, they redeploy the master branch to production without worry of hotfix, rollbacks, deploy up to a specific commit, or redeploy the artifacts from the last release. After the deployment is deemed a success, then the feature branch is merged with master.

That sounds promising, but I'm missing how they have confidence that what is merged with master is good. Sure, I'm assuming they merge from master to their branch before deployment and certainly before merging the branch to master. But the feature branch is what was tested, not what was in master. Maybe I'm thinking too old school with merge conflicts with app.config/web.config files and such. I'm left with thinking they have this covered, but with trepidation.

GitHub uses the Hubot chat framework custom designed by GitHub for Github. The chatbot performs the build/release/deploy process. It locks environments down such as prod to prevent developers from stomping on each other. It performs Continuous Integration (CI) checks to verify you've merged master to your branch and all tests are passing. But wait there's more! Hubot monitors the performance of the site and other apps and will create issues in GitHub or post in chatrooms. Hubot supports extensions to allow developers to add new features.

Christian did a fine job in the presentation and answered questions well. However, I've come out of the meeting with mixed feelings. I went in already knowing the basics of what Hubot can do from a previous demo or article I read. The demo provided inspiration of what we may be able to do one day with chatbots, but we don't have access to Hubot.

As a GitHub noob, I was looking more for how to create a build/release pipeline with GitHub with the tools available today. Webhooks sound like the right way to go, but it was glossed over and that's what I wanted to learn more about. In the end, I suppose I'll personally continue to use VSTS to create Build/Release pipelines where I can select from my VSTS or GitHub repositories.

# Introducing Domain-Driven Design – by Steve Smith

**You will learn: Domain-Driven Design fundamentals and Repository design pattern**

[Slideshow](https://www.slideshare.net/ardalis/introducing-domain-driven-design-dogfood-con-2018?qid=4cb0aa9c-a55c-4354-86ac-5b48f6b6409c&v=&b=&from_search=5)

Steve Smith and Julie Lerman published an excellent [Domain-Driven Design Fundamentals course on PluralSight](https://www.pluralsight.com/courses/domain-driven-design-fundamentals). He also created a [Domain-Driven Design Guestbook on GitHub](https://github.com/ardalis/ddd-guestbook) as an example implementation. There are a couple podcasts on the subject in his [Weekly Dev Tips](http://www.weeklydevtips.com/).

Domain-Driven Design (DDD) is all about focusing on the problem domain by providing tools for better communication, applying clean code standards, and following principles and patterns to solve difficult problems. Here are some benefits of following DDD.

* Flexible
* Customer's vision/perspective of the problem
* Well organized and easily tested code
* Business logic lives in one place
* Many great patterns to leverage

Warning: DDD should only be applied to complex designs. Here are some reasons.

* Time and effort
* Learning Curve
* Team/Company Buy-in to DDD

DDD concepts

* Core Domain: company core skill
* Problem Domain: specific problem your app will address/solve
* Generic SubDomain: apps/features you interact with
* Bounded Context: A logical boundary for a model so it only knows and does what is necessary within the context. Example: A Client model in billing may require contact information whereas scheduling may only require an email address to notify the client. Don't pollute the model with properties you don't need.
* Ubiquitous Language: same terms as business

This was learned within the first fifteen minutes. Please watch his PluralSight course for more. Totally worth it to better understand DDD.

As typical, Steve filtered down a very complex subject into its basic components. His speech is evenly paced and he packs a lot of information in his sessions. He wastes no time and usually goes over while still answering a few questions that come up. Having already watched the DDD PluralSight course and reading other articles, this session was more of a refresher to make sure I still had a good grasp of the concepts. One point I'd make is that the concepts of following clean architecture, reducing dependencies, and repositories are solid techniques that can be used even if you aren't following DDD.

# Addition by Abstraction: A conversation about abstraction and dynamic programming – by Jerren Every

**You will learn: Gerkin, YAML, Abstracting the data from the test code for reuse, a few refactoring techniques, and TDD basics**

I could not find the slideshows at the writing of this review. [Reference](https://www.pnsqc.org/addition-by-abstraction-a-conversation-about-adding-value-to-an-automated-test-suite-though-dynamic-coding-and-abstraction-concepts/)

This session targeted Gerkin, YAML, etc. All of these principles can be applied to other test frameworks such as C# .NET unit tests to create integration techniques.

By following principles of [orthogonality](https://en.wikipedia.org/wiki/Orthogonality_(programming)) your code and tests should be as simple as possible and do just one thing. This means if you have bad code to update your few tests should cover it since the changes are so isolated.

Step Definition > Helper method > Service being tested

* Step definitions request test info from Data Helpers that gathers all of the info needed for a test.
* The Helper method accepts the gathered test info and performs an action.
  + This allows the test to be reused for other tasks by passing new test info.
  + Helper modules are like Page Objects that abstract away logic from data.

There are several common [refactoring](https://refactoring.com/) techniques out of the available [catalog](https://refactoring.com/catalog/) that help. Ex. [Extract Class](https://refactoring.com/catalog/extractClass.html), Pull Up [Field](https://refactoring.com/catalog/pullUpField.html) / [Method](https://refactoring.com/catalog/pullUpMethod.html), and Rename Class / [Method](https://refactoring.com/catalog/renameMethod.html). Refactoring should not be forced and should feel nature.

[Test-Driven Development](https://en.wikipedia.org/wiki/Test-driven_development) follows this process. Red-Green-Refactor: Add a test that fails, make it work, refactor with confidence that your tests will still pass.

* Add a test
* Run all tests and see if the new test fails
* Write the code
* Run tests
* Refactor code
* Repeat

Don't worry about full code coverage. As an example don't worry about testing getters/setters. Avoid dynamic failures where you catch an error and retry as there are typically better ways of handling the scenario. Standard errors should be handled, only the exceptional errors should be thrown as exceptions. Test positive (expect a calculation to evaluate to a specific value) and negative (if an exception can be thrown, then you should test it) cases.

Jerren presented well for the most part. He tried to get feedback from the audience to make the session more interactive or conversational as the title implies. Most of the audience just waited a few moments before he continued. He knew the topic very well. I would recommend cutting out some of the attempted audience conversation and add a few more nuggets of information.

# From Zero to Serverless – by Chad Green

**You will learn: An introduction to serverless, create an Azure Function in-portal and in VS2017**

[Slides](https://www.slideshare.net/chadgreen/from-zero-to-serverless-dogfoodcon-2018)how

The first fifteen minutes of this session was an overview of how IT has progressed from servers to serverless (still servers performing the processing, you just don't maintain them). Chad also gave a high level overview of some Azure features such as logic apps. Given the nature of the session I understand why Chad did the introduction. If you've read my reviews of sessions in the past I prefer skipping these types of introductions to get to the heart of the subject matter.

Create an Azure Function in <https://portal.azure.com>

1. In Azure > Create Resource > Compute > Function App
   1. Use the consumption price plan unless you already pay for an App Service plan
   2. Functions require storage, so create or use an existing storage account.
2. Open the function app > click New Function
   1. To get one up and running quickly use In-portal.
   2. Later you'll want to use Visual Studio so you can test locally at no cost in the VS2017 (15.6.6 or higher) with Azure Development tools installed. This way you may also check the function into VSTS/GitHub and create a build and release definition to deploy the function to Azure. Those steps were learned in the Friday session.
3. Choose More > HttpTrigger
   1. The function may be run immediately to display a hello world message where the name is populated from the post.

Chad recommended not using Api Management, but I didn't catch the reasons in my notes. I believe it's due to the premium costs to support enterprise production as mentioned in his last session below.

While functions may be deployed together with other functions and resources he recommends keeping each separate so only what is changed is deployed. Remember each function has its own endpoint and is considered a web service.

Proxies provide more control over functions such as url rewriting to create routes to a cleaner REST based url.

Functions should:

* Do one thing
* Be [Idempotent](https://en.wikipedia.org/wiki/Idempotence)
* Should finish as quickly as possible

General best practices:

* Avoid long running functions
* Handle cross function communication
* Write to be stateless
* Write defensive functions

Scalability best practices:

* Do not mix test and prod code in the same function
* Use async code but avoid blocking calls
* Receive messages in batch whenever possible
* Configure host behaviors to better handle concurrency
* Start small, replace 1 API or background processing at a time

Logic apps are orchestrators, so use them to pass info between functions.

Chad recommended creating all of your business logic in NuGet packages that are unit testable, so the function's only job is to call the main process exposed by the NuGet package. This approach is beneficial when the main process is also used by other applications on tablets, phones, pcs, etc.

Another approach is to keep those projects in the same solution as the function including the unit tests. As long as it's not needed elsewhere, this keeps everything together. Chad mentioned that he could not properly unit test the actual function, so more research to be done. At least by isolating the bulk of the process to another project, that part can be unit tested.

My main goal for attending this session was to learn how to create an Azure Function and they were met. As mentioned earlier, the session could have included so much more without the intro. However, Chad did a great job presenting the process of creating an Azure Function with pros/cons/tips, so in the end all expectations were met.

# Friday Keynote – Building Effective Enterprise Software Communities – by Christian Weber

Building upon Nicole Jackson's keynote, Christian emphasized the need for the team/community to support and encourage communication and contribution by the team members. He related an experience where he spent months working on a project that he later found out was almost identical to another project another team had already completed before he even started and could have been used as a template to reduce time and effort. If only they had communicated. Build your reputation over time as someone dependable, delivers, contributes to the team, and can be trusted. Trust empowers confidence to do your best and to prove you've earned that trust.

# Baseball, Actors, and Bots – by David Hoerster

**You will learn: C# Akka.net (an event driven actor system), Luis Bot, and SignalR**

The presentation's slideshow was not available, so here's his [GitHub](https://github.com/DavidHoerster?tab=repositories).

The [Actor Model](http://getakka.net/articles/intro/what-are-actors.html) is a system of small units of concurrent computation. Each unit is an actor that communicates to each other via messages. Actors can have children that can be supervised. Actors can be microservices and can be treated as bounded context (DDD).

Actors are a light weight class that encapsulates state and behavior. State can be persisted and behavior can be switched. Actor's have a mailbox to receive messages and will process them like a queue. Actor's are thread safe. They have a lifecycle and are garbage collected.

[Akka.NET](http://getakka.net/) is an Actor framework. Everything starts from an ActorSystem which is costly so create it once or just a few. Actors are created in a [hierarchy supervisory](https://doc.akka.io/docs/akka/2.5/general/supervision.html). Each actor has its own uri (like a WebApi).

Sending messages should be immutable since other actors that receive the message would not know about the changes. Actor's should Tell other actors what to do. Actor's should rarely if ever Ask other actors what to do. Sending messages is asynchronous. Even though actor's are async, the requests are processed in a queue, but only for that actor. Child actors and parent actors are async to each other.

[Luis.ai](https://www.luis.ai) was only briefly mentioned at the end of the session, but he gave a nice high level overview of how utterances, intents, and entities work. For more on Luis.ai, check out the [Dog Food Conference 2017 review - Rise of the Bot: Building Interactive Bots with Language Understanding By Brian Sherwin](https://www.danylkoweb.com/Blog/review-dog-food-conference-2017-L4).

David tied the Actor system with [SignalR](https://www.asp.net/signalr) to display Baseball stats. As the system was processing stats from a database supplying the info, the page updated constantly. Opening info for players and teams called the actors associated with each and displayed their stat blocks and trading cards. A quick run through a chatbot using the Luis.ai gave another method to access the actors. I thought it was pretty cool how it all came together.

As we work more with web services, microservices, and serverless logic apps/functions, the Actor system is an interesting way to visualize the concepts. Overall, I think many developers would benefit from attending this session to understanding the Actor system architecture even if they want to roll a simpler roll your own version instead of using Akka.Net.

David did an excellent job explaining Akka.NET and the actor system. Probably would have liked more on the Luis bot since "bots" in the title caught my attention the most. The live portion of the website involved SignalR and it was barely mentioned.

Perhaps cutting some of the intro to make more room for the bot and SignalR would help, but it's fairly packed full of info. This was tough, because I really liked David's session as it is, I just want more info on the integration pieces with the Luis bot and SignalR. Hmm, maybe making them a session on their own would make the end feel less rushed.

# Dipping your toe into Cloud Development with Azure Functions – by Brian T. Jackett

**You will learn: Azure Functions, storage, security, durable functions, and other interactions with Azure features**

[Slideshow](https://www.slideshare.net/BrianTJackett/cloud-dev-with-azure-functions-dogfoodcon-2018-brian-t-jackett?qid=da12d95b-9f8e-4cd6-b75b-efe3b037cd5e&v=&b=&from_search=1)

Brian was direct and to the point. He started his presentation and we were immediately overwhelmed in a knowledge dump few expected. Totally worth it!

In Azure, you can heart a service to bubble it to the top when searching.

[Functions as a Service (FaaS)](https://en.wikipedia.org/wiki/Function_as_a_service) is code that runs in response to a triggered event. Only one triggered event. They have [input bindings and output bindings](https://docs.microsoft.com/en-us/azure/azure-functions/functions-triggers-bindings). [Event grids](https://azure.microsoft.com/en-us/services/event-grid/) and [event hubs](https://azure.microsoft.com/en-us/services/event-hubs/) act as an intermediary between Azure Functions and other services that can't trigger functions.

Azure Functions can be precompiled (ex. Visual Studio) and uncompiled (ex. VS Code, Azure Portal). Functions in the cloud and on Premises are the same.

It is highly recommended to use [Azure Storage Explorer](https://azure.microsoft.com/en-us/features/storage-explorer/) and [Service Bus Explorer](https://github.com/paolosalvatori/ServiceBusExplorer). I actually used the Storage Explorer recently and it works pretty well to explore the Azure Storage resources. Use [Postman](https://www.getpostman.com/)/[Fiddler](https://www.telerik.com/fiddler)/etc. to test the Function. I've also used [SoapUI](https://www.soapui.org/). New Functions should be in the same region as the storage.

Extensions:

* VS2017: Azure Functions and WebJobs Tools
  + Make sure you keep the extension up-to-date. I worked on a colleague's Azure Function and immediately ran into errors until I updated to the latest extension.
* VSCode: Azure Functions

Functions support tokenization. In the local.Settings.json add token declarations even in Attributes [%token%]. Enable CORS to allow your domain to call the Function. Function > Platform Features > CORS > Add your domain to the whitelist.

Take advantage of the dozens of [cloud](https://docs.microsoft.com/en-us/azure/architecture/patterns/) [design](http://www.cloudpatterns.org/) [patterns](https://azureinteractives.azurewebsites.net/CloudDesignPatterns/default.html) already defined including a great PluralSight course I recommend: [Cloud Design Patterns for Azure: Availability and Resilience by Barry Luibregts](https://app.pluralsight.com/library/courses/azure-design-patterns-availability-resilience/table-of-contents).

Azure Functions are treated as event driven programming, do not do stateful. Design to run as fast as possible with the smallest footprint. Minimize dependencies such as NuGet packages.

For Authentication use the Active Directory application with fine grained OAuth permissions. Authenticate with Client ID/Password and Certification. For more information consult his post on [Azure Functions Calling Azure AD Application with Certificate Authentication](https://briantjackett.com/2018/07/25/azure-functions-calling-azure-ad-application-with-certificate-authentication/).

Store credentials in the Managed Service Identity where you can assign permissions to key vault, storage, etc. App Settings are not very secure since you can see clear passwords in the portal. [Azure Key Vault](https://azure.microsoft.com/en-us/services/key-vault/) is very secure. Make sure you configure authentication correctly, disable anonymous, and disable the default Function home page. Use [Shared Access Signature (SAS)](https://docs.microsoft.com/en-us/azure/storage/common/storage-dotnet-shared-access-signature-part-1) tokens when possible.

[Durable Functions](https://docs.microsoft.com/en-us/azure/azure-functions/durable-functions-overview) define workflow in code. Without durable functions if there is an error between communication queues and functions, it can be very difficult to track down and finish the process. A single orchestrated flow can respond to errors, retries, etc. at any point. Other durable functions patterns include: Functional Chaining, Fan-out / Fan-in, and Human interaction.

Brian is a great speaker. He went through a ton of information very quickly and thoroughly. I prefer these kinds of sessions. Posting slideshows helps review any points that may have been missed. This session reminded me a lot of a Steve Smith (ardalis) session, so from me that's an esteemed compliment.

# Microsoft Teams – The Missing Manual – by Ricardo Wilkins

**You will learn: Microsoft Teams (nailed it!)**

[Slideshow](https://www.slideshare.net/ricardo303/microsoft-teams-the-missing-manual?qid=1d9ac6a0-348e-461b-947f-788b96fafecf&v=&b=&from_search=10)

[Teams](https://products.office.com/en-us/microsoft-teams/group-chat-software) is a collection of people, content, and tools surrounding different projects. Teams is powered by SharePoint Online.

Channels

* Dedicated sections within a team to keep conversations organized.
* Place where everyone on the team can have open conversations.
* Can be extended with tabs, connectors, and bots.

If a conversation in a channel gets chatty, create a channel and add a reference link back to the original post. Chatty in this case means a sub-topic that may need more ongoing conversation. If you plan to add documents, consider adding a channel.

When a project is done, do not delete it, just unfavorite the channel so it's not in your active list.

[SharePoint Online Search](https://docs.microsoft.com/en-us/sharepoint/overview-of-search) across the enterprise allows you to do a Bing search and if you are in your company's network, the company's Office365 products will be searched as well included at the top of the results. Of course this will only happen after setup by an administrator. While you could use [Delve](https://support.office.com/en-us/article/what-is-office-delve-1315665a-c6af-4409-a28d-49f8916878ca), it does not currently work with Teams.

Add titles to conversations. If someone wants you to be aware they can mention you in a conversation with @username or @teamname and you'll receive an email notifying you or your team was mentioned in a conversation. @General only notifies those in the team project

Private chats with 3 or more can have group names to make it clearer what the discussion is about. Follow only teams and channels you are interested in.

The activity section references all conversations happening for channels you have favorited. The search bar allows for commands by typing / and the command.

SharePoint files must be dropped in the SharePoint Teams folder or it won't show up in Teams. Otherwise just add through Teams > Team > Files tab.

You can add bookmarks to tabs to websites, file references, and many other things.

Add conversations on files will also add the conversation to the channel the file is in, so others are aware of the conversation and may contribute.

Create a Team of 1 as a personal notebook. Add a One Note tab to help you organize your thoughts instead of just typing conversations to yourself.

My team does not use Teams effectively. We're all noobs who have been using it for less than a year, so I figured I'd check out this session to see if there is room for improvement. Here are our current issues with Teams.

* We don't always reply under the conversation we intend, typically creating a new conversation. We'd like to drag and drop the reply to the appropriate conversation or have an ellipsis with an option to move. Ricardo says this happens frequently and users should copy the content, reply to the correct conversation, paste, and then delete the incorrect entry.
* We've added many discussions regarding projects as channels under a single Team project. Ricardo's demo showed that perhaps we should have a separate Team project for each project with only the users who need access added to the team. Cool, so while I can start the new Team project, we can't move channels between Team projects.
* We can't move conversations between channels, so you have a couple options. Manually copy all of the contents as new conversations in the other channel and then delete the original conversation. Another option is to create a new conversation in the other channel and then add reference links to those conversations.
* Since we can't move channels/conversations and we don't want to delete them we'd like to reference the channel/conversation, but otherwise it should be archived and no longer seen in the channel. The best option here is unfavorite the channel, so it doesn't show up in the favorites.
* We're looking for how we want to manage a Wiki. Teams has a Wiki, but should be ignored in favor of using the Sharepoint Wiki. We haven't had much luck with the older Sharepoint Wiki search results, because we couldn't filter the results properly. The search flooded with irrelevant documents, when we were only interested in the documents under IT and not business or other parts of the organization. Ricardo said we should be able to filter down, so I'll work with our SharePoint administrator to see if we can do better.

Teams has limitations for what I'm trying to do, but now I have best practices on how to handle the issues mentioned above even if I'm not satisfied with them. Hopefully, some of these concerns will be addressed in future updates of Teams. Ricardo knew the content and handled questions, including many of mine, very well. Given this was more of a product review, I was fine with the standard presentation style.

# Building an Ultra-Scalable API Using Azure Functions Without Too Much Worry – by Chad Green

**You will learn: Deploying Azure Functions using Visual Studio Team Services**

The presentation slideshow was not available at this time. [Deploying Azure Functions using Visual Studio Team Services](https://docs.microsoft.com/en-us/labs/devops/deployazurefunctionswithvsts/) covers most of the build/release pipeline.

The first session was about creating the Azure Function and this session was about deploying that Azure Function. This was the last piece needed to understand best practices from real world examples.

Unfortunately, the first 15 minutes of this presentation was a repeat of the first 15 minutes from the previous presentation. I came to this session as a follow-up/continuation of the previous presentation as it was billed in the first session. I heard all of this already and was tempted to leave to catch something new. Thankfully I didn't.

A couple new pieces of info were brought up not mentioned before as I'll note shortly. However, the next 15 minutes described the problem that needed solved and why the deployment by region architecture was chosen. His session description was upfront about both the introduction and this scenario build up, so while it wasn't unexpected, this wasn't what I came for and was tempted to leave to catch something new. Again, thankfully I didn't.

Basic principles to follow

* Stateless
* Coarse grained API
* Embrace failure
* Avoid instance specific configurations
* Simple automated deployment
* Monitoring
* KISS – variation: Keep It Small and Simple

Design Goals

* Distribute API Development
* Support for multiple languages
* Minimize Latency
* …

In this scenario, the [Azure Function needs deployed to multiple regions](https://docs.microsoft.com/en-us/azure/machine-learning/studio/how-to-deploy-to-multiple-regions) each with their own storage account in each region. Most of the locations dependent upon these functions have minimal Wi-Fi availability with poor signal strength, so anything that can be done to speed up the process the better. Chad recommended using the [Traffic](https://azure.microsoft.com/en-us/services/traffic-manager/) [Manager](https://docs.microsoft.com/en-us/azure/traffic-manager/traffic-manager-overview) to route requests to the closest region to solve this issue.

Be aware that API Management tools premium to support enterprise production [costs are expensive to support multiple regions](https://azure.microsoft.com/en-us/pricing/details/api-management/).

I love adding mockable unit tests to my application, but Chad said he had problems mocking the main Azure function. Instead he isolated the functionality to a NuGet package that could be unit tested. He tested Entity Framework (EF) by using an in memory database. I like hiding EF behind a repository to further separate concerns.

The entire connection string is stored in the Key Vault, so no one has access to the credentials. He generated an ARM template to populate the base of the connection string.

Naming convention: AppName-Entity-Version-AzureRegion[-Environment]

* Ex: PHF-EmployeeType-V1-USE2-DEV

The Build definition followed the standard deployment pattern. Only continue to the next task if the previous task succeeded.

1. Get NuGet packages from [nuget.org](https://www.nuget.org/) and the [Azure Artifacts](https://docs.microsoft.com/en-us/vsts/package/overview?view=vsts) [Package Management](https://www.dotnetcurry.com/visualstudio/1408/nuget-package-management-vsts)
2. Build the application (Azure Function)
3. Run tests
   1. In this case all of the unit tests are run when building the NuGet package for the Azure Function in a separate build definition.
4. Publish [Postman](https://www.getpostman.com/) tests
   1. The tests are checked in with the source code under a separate folder
5. Publish Artifacts

Use PowerShell to create Traffic Manager. Create Azure Resources with an ARM template overriding the template parameters.

The Release definition followed this pattern. Only continue to the next task if the previous task succeeded.

1. Disable Endpoints
2. Deploy Azure Function to testing slot
3. Install [newman](https://www.npmjs.com/package/newman) package (npm install newman)
   1. Used by Postman
4. Replace tokens in the json files
5. npm run newman
6. Publish integration test results
   1. They should appear in the Release Test tab
7. Swap test slot with production slot
8. Smoke web test
9. Enable Traffic Manager

The tasks were cloned for each region for each environment with preconditions that the next environment could not be deployed to until the previous deployment was successfully deployed.

Traffic Manager routing method

* Performance
* Weighted
* Priority
* Geographic

Here's an expensive mistake that we can avoid. The obvious routing method should be Performance since that is our goal. However, it calls each function in each region frequently with a smoke test. This allows the traffic manager to know which region should be used when an actual request comes. As you can imagine the costs piled up and while a large company can absorb these costs in favor of performance, it does not make since for smaller companies. Since the app is already deployed to the most hit regions, choosing Geographic makes the most sense in this setup.

Looking back on the session, I would recommend removing the first half hour. Start with explaining how the DevOps pipeline works at a high level for a few minutes and immediately show a basic build definition just like the one in the presentation. Next show a basic release definition that deploys to a single region. I think the immediate jump to multiple regions threw some off. Then explain in a couple minutes that we now need to scale up to deploy to multiple regions and show the release definition showed in the presentation. Now you've shown how to scale up where in the presentation we started scaled up. I would also recommend adding in highlights of the info discussed during the half hour after the session ended which mostly centered on the release definition preconditions, approval process, and other clarifications.

In the end I gathered all of the desired information expected from the session and it's after school special. I was grateful that Chad took the time afterwards to answer all of my questions as well as those from another session goer trying to determine if going the route of Azure Functions is the right decision.

# Conclusion

Speaking with others, I'm reminded that my expectations of a session and others differ. They may value the introductions used to help convince them to buy into a concept or product. For me, if I choose to go into a session, I've already bought into it enough. I want to know all there is within the short time we have, so I get the gist and can further research the subject on my own afterwards or put it in my back pocket for later. That's why I recommend sessions add a clearer "You will learn" highlight to help set expectations. Session goers could then preemptively review the topic to see if they want to learn more and be sold on all of the benefits of the proposed solution.

My hopes are that this type of review while extensive and fairly detailed has helped others decide to research these topics. For the presenters, I pose that my expectations are my expectations. Please take them as advice and positive criticism, and please also get others opinions.

I enjoyed Dog Food Con 2018 thoroughly and came away with loads of new tools and techniques to try, especially in the realms of Azure Functions. The sessions I attended this year met my expectations very well. What session was your favorite? Did you attend any sessions that felt more like a marketing pitch? Do you agree with adding a "You will learn" highlight? Or perhaps session titles and descriptions should be clearer about what will be covered?