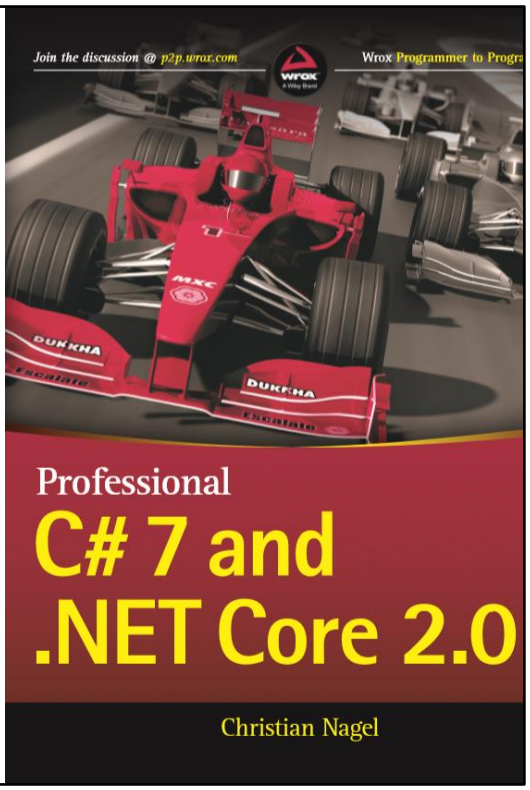
Web APIs with .NET

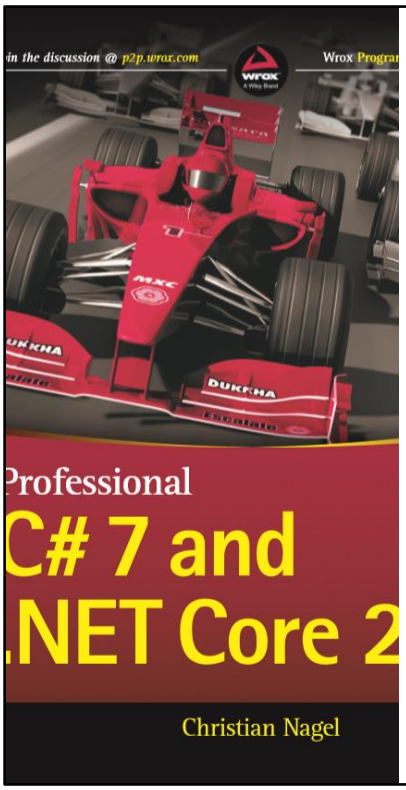
„by Christian Nagel“ :
{ „CN innovation“ }

CN innovation - Christian Nagel

- Training
- Coaching
- Coding
- Writing

- csharp.christiannagel.com
- www.cninnovation.com
- @christiannagel
- Microsoft MVP (Visual Studio)





Course Materials

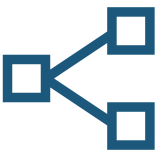
- The Book
  - Professional C# 7
- Slides
- Code Samples and Labs
  - https://github.com/cnilearn/

Topics

- .NET Core
- Dependency Injection
- REST APIs
- Web APIs with ASP.NET Core
- EF Core
- Swagger und OpenAPI
- Authentication
- Azure Functions
- SignalR
- Azure SignalR Service
- Workers
- GRPC
- Microservices
- Docker

Introduction

Experience

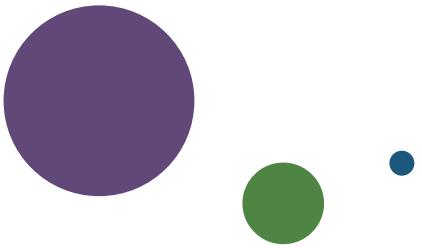


Expectations

Sample Code

get sample code

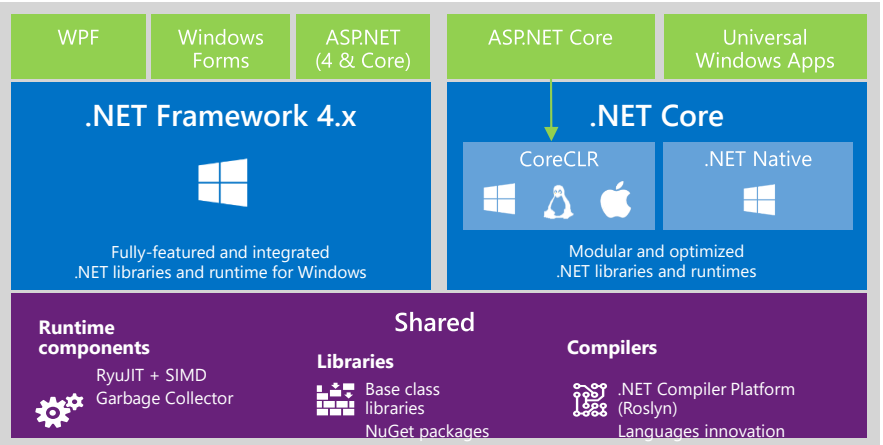
https://www.github.com/cnilearn



.NET CORE & ASP.NET CORE INTRO

Chapter 1 & 20 & 29 – Professional C# 7

.NET - Overview



.NET Core Support

Version	Release Date	Patch Version	Support Level	Support End
6.0	Nov-2021	NA	LTS	Nov-2024
5.0	Nov-2020	5.0.7	Current	Feb-2022
3.1	Nov-2019	3.1.16	LTS	Dec 3, 2022
~~3.0~~	~~Sep, 2019~~	~~3.0.3~~	~~Current~~	~~Mar 3, 2020~~
~~2.2~~	~~Dec 4, 2018~~	~~2.2.8~~	~~Current~~	~~Dec 23, 2019~~
2.1	May 30, 2018	2.1.28	LTS	Aug 21, 2021
~~2.0~~	~~Aug 14, 2017~~	~~2.0.9~~	~~Current~~	~~Oct 1, 2018~~
~~1.1~~	~~Nov 16, 2016~~	~~1.1.13~~	~~LTS*~~	~~Jun 27, 2019~~
~~1.0~~	~~Jun 27, 2016~~	~~1.0.16~~	~~LTS~~	~~Jun 27, 2019~~

https://dotnet.microsoft.com/download/dotnet-core

dotnet Tools (.NET Core CLI)

- dotnet tool
  - install tools
- dotnet new
  - initialize sample C# console app
- dotnet restore
  - restore dependencies
- dotnet build
  - builds a .NET Core application
- dotnet publish
  - publishes a .NET portable or self-contained application
- dotnet run
  - runs the application
- dotnet test
  - runs test using a test runner
- dotnet pack
  - creates a NuGet package

.NET CORE FOUNDATIONS

Topics

- Dependency Injection
- Configuration
- Logging

Dependency Injection

- Dependency Injection
  - Hollywood Principle
  - Inversion of Control
  - Injection is passing of a dependency to a dependent object

Without Dependency Injection

```
public class SampleController
{
  private readonly FooService _service = new();

  public void Action() =>
    _service.Foo();
}
```

With Dependency Injection

```
public class SampleController
{
  private readonly IFooService _service;

  public SampleController(IFooService service) =>
    _service = service;

  public void Action() =>
    _service.Foo();
}
```

Dependency Injection Container

Register all services and contracts in one place

Singleton, transient, or scoped type, or passing a factory method

Dependency Injection Container

- Without Container

```
SampleController controller = new(new FooService());
controller.Foo();
```

- With Container

```
var controller = Container.GetService<SampleController>();
controller.Foo();
```

Dependency Injection Summary

- Advantages
  - Unit Tests
  - Platform Independence

Configuration

- Configuration for Development, Production, Staging…
- Visual Studio had XML Transformations
- Now we have an easier solution

.NET Core Configuration

- Packages Microsof.Extensions.Configuration.*
- Flexible Configuration
- JSON, XML, INI, Arguments…
- User Secrets

- Types:
  - IConfigurationBuilder
  - IConfigurationRoot

Configuration Summary

- Flexible
  - JSON, INI, XML, Environmental Variables
  - User Secrets

Logging

- Generic Interfaces for Logging
- ILogger
  - Scope, Log, LogLevel
- ILoggerFactory
  - Add providers
  - create loggers

Host

- .NET Core 3.x
- Common Host for all .NET Core Applications
- Configure Services, Logging, Configuration…

Summary

Dependency Injection

Configuration

Logging

ASP.NET CORE WEB API

Chapter 32 – Professional C# 7

Topics

Overview

Routing

Formats and Binding

REST
Representational
State Transfer

- Client-Server
- Statelessness
- Cachability
- Layered System
- Code on demand (optional)
- Uniform interface

Uniform Interface

- Resource identification in requests
- Resource manipulation through representations
- Self-descriptive messages
- Hypermedia as the engine of application state (HATEOAS)

REST Levels

Level 0 - HTTP

- One URI, one HTTP method
- XML-RPC, most SOAP services

Level 1 - Resources

- Many URIs, one HTTP method
- Most "RESTful" services

Level 2 – HTTP Verbs

- Many URIs, multiple HTTP methods
- Microsoft Azure

Level 3 - Hypermedia

- Resources describe capabilities and links
- www. AtomPub, Netflix, LaunchPad

Leonard Richardson, https://www.crummy.com/writing/speaking/2008-QCon/act3.html

Web API

- Dispatches Actions based on HTTP verbs
- Content – XML, JSON, PDF, VCARDs…
- Content type negotiation

REST Results and Status Codes

HTTP Method	Description	Request Body	Response Body
GET	Returns a resource	Empty	The resource
POST	Adds a resource	The resource to add	The resource
PUT	Updates a resource	The resource to update	None
DELETE	Deletes a resource	Empty	Empty

HTTP Status Code	Controller Method	Type
200 OK	Ok	OkResult
201 Created	CreatedAtRoute	CreatedAtRouteResult
204 No Content	NoContent	NoContentResult
400 Bad Request	BadRequest	BadRequestResult
401 Unauthorized	Unauthorized	UnauthorizedResult
404 Not Found	NotFound	NotFoundResult
Any status code		StatusCodeResult

Status Codes

Controller

- Route
- Dependency Injection

```
[Produces("application/json", "application/xml")]
[Route("api/[controller]")]
public class BookController : ControllerBase
{
  private readonly IBooksService _service;
  public BookController(IBooksService service)
  {
    _service = service ?? throw new ArgumentNullException(nameof(service));
  }
```

Return Books / a Book

- GET

```
[HttpGet]
public IEnumerable<Book> GetBooks() => _service.GetBooks();
```

```
[HttpGet("{id}", Name = nameof(GetBookById))]
[ProducesResponseType(404)]
public ActionResult<Book> GetBookById(int id)
{
  Book book = _service.Find(id);
  if (book == null) return NotFound();
  else
  {
    return Ok(book);
  }
}
```

ActionResult<T>

- Return status code or data
- Since .NET Core 2.1

Add a Book

- POST

```
[HttpPost]
[ProducesResponseType(201)]
public IActionResult PostBook([FromBody] Book book)
{
  if (book == null)
    return BadRequest();

  _service.Add(book);
  return CreatedAtRoute(nameof(GetBookById), new { id = book.Id });
}
```

Update a Book

- PUT

```
[HttpPut("{id}"]
[ProducesResponseType(204)]
[ProducesResponseType(400)]
[ProducesResponseType(404)]
public IActionResult PuBook(int id, [FromBody] Book book)
{
  if (book == null || id != book.Id)
    return BadRequest();
  if (_service.Find(id) == null)
    return NotFound();

  _service.Update(book);
  return new NoContentResult();
}
```

.NET Client

- Use HttpClient

```
public void SendRequest(HttpClient client)
{
  client.BaseAddress = _baseAddress;
  HttpResponseMessage resp = await client.GetAsync(requestUri);
  resp.EnsureSuccessStatusCode();
  string json = await resp.Content.ReadAsStringAsync();
}
```

JSON Serializer

- System.Text.Json
- New since .NET Core 3.1

```
JsonSerializer.Serialize(obj);
```

XML

- Server: add XML formatter

```
services.AddControllers(
    options => options.RespectBrowserAcceptHeader = true)
    .AddXmlSerializerFormatters();
```

- Client: accept application/xml Format

```
client.DefaultRequestHeaders.Accept.Add(
  new MediaTypeWithQualityHeaderValue(„application/xml“));
```

Metadata with Web API

- Use Swagger
  - http://swagger.io
- A new Standard: OpenAPI
  - http://openapis.org
- Compare this with WSDL & SOAP Web Services
- ApiController attribute –improve metadata for the controller

Analyzers

- Microsoft.AspNetCore.Mvc.Api.Analyzers
- analyze web APIs, check for missing attributes

Web API Checklist

https://mathieu.fenniak.net/the-api-checklist/

- Idempotent methods
- Authentication
- 201 Created, 202 Accepted
- Design for intent
- Versioning
- Bulk operations
- Pagination
- …

Microsoft API Guidelines

- https://github.com/microsoft/api-guidelines/blob/vNext/Guidelines.md
- Taxonomy
- Client Guidance
- Consistency
- CORS
- Collections
- Delta Queries
- Versioning
- Long Running Operations
- Throttling, Quotas, Limits
- Push Notifications
- Naming

Summary

REST Style

Routing

Actions

Resources

Guidelines

MODEL

Chapter 26 –
Professional C# 7

EF Core

- Create a Model
- Create a DbContext
- Specify a Provider
  - using OnConfiguring
  - using Dependency Injection

Data Annotations

Table

map to a database table

Column

map to a column

TypeName – specify data types

MaxLength

Fluent API for Models

- Alternative to data annotations
- More features
- Override OnModelCreating with the context

Migrations

Maintain database schema and code changes

Allows to update the database from code or SQL Script

Extension with dotnet tool:

dotnet ef

Model Binding

- Model binder builds paramters
- Finds all Album Properties for Binding

```
[HttpPost]
public ActionResult Edit(Album album)
{
  // ...
}
```

- Uses the name of the paramter

```
public ActionResult Edit(int id)
{
  // ...
}
```

Explicit Model Binding

- Without action parameters needed

```
[HttpPost]
public ActionResult Edit()
{
  var album = new Album();
  if (await TryUpdateModelAsync(album)
  {
    db.Entry(album).State = EntityState.Modified;
    db.SaveChanges();
    return RedirectToAction("Index");
  }
  else
  {
    // return view data to the view as needed to resolve
    return View(album);
  }
}
```

Summary

- Entity Framework Core
- Migrations
- Using EF Core with ASP.NET Core

.NET CLIENT

HttpClient

- Async Methods
- GET, POST, PUT, DELETE…

JSON String Issues

PAY ATTENTION TO LONG STRINGS

USE STREAMS INSTEAD

HttpClientFactory

- Dispose issue
  - Socket can stay alive for a timeout
  - Running out of sockets
- Use Factory for HttpClient
- Inject HttpClient or factory

Retries

- Using Polly and HttpClientFactory

Summary

HttpClient

HttpClientFactory

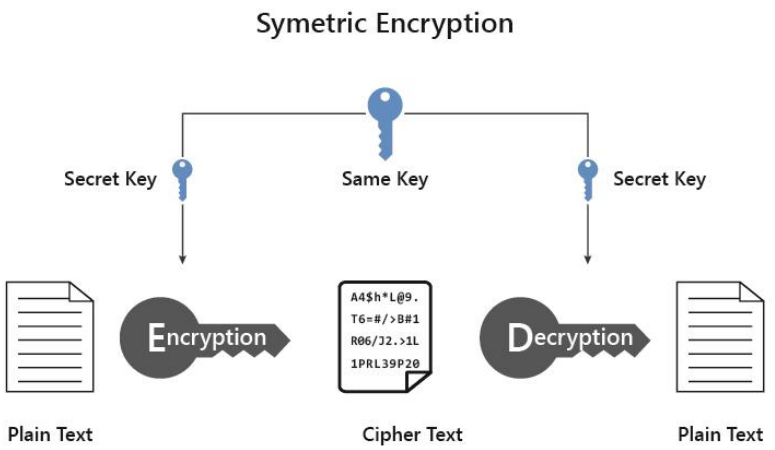
Polly

SECURITY

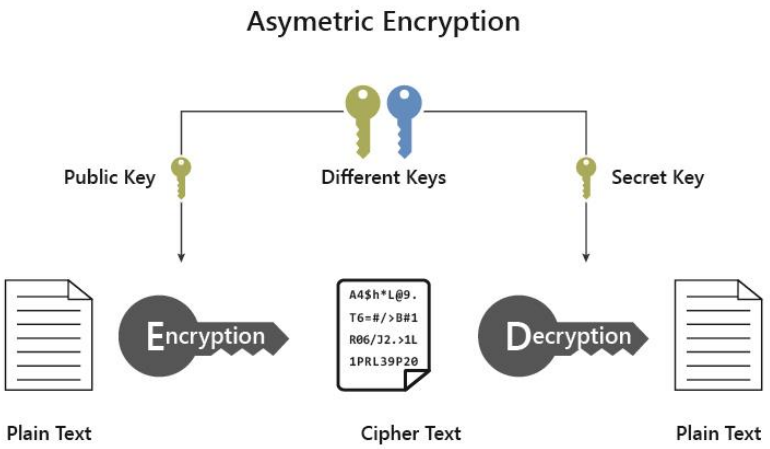
Encryption

- Symmetric encryption
  - Same key for encryption and decryption
  - Fast, commonly used, easy to implement
- Asymmetric encryption
  - The sender uses the public key to encrypt the information
  - The receiver keeps the private key secure
  - Only the receiver can decrypt the information by using the private key

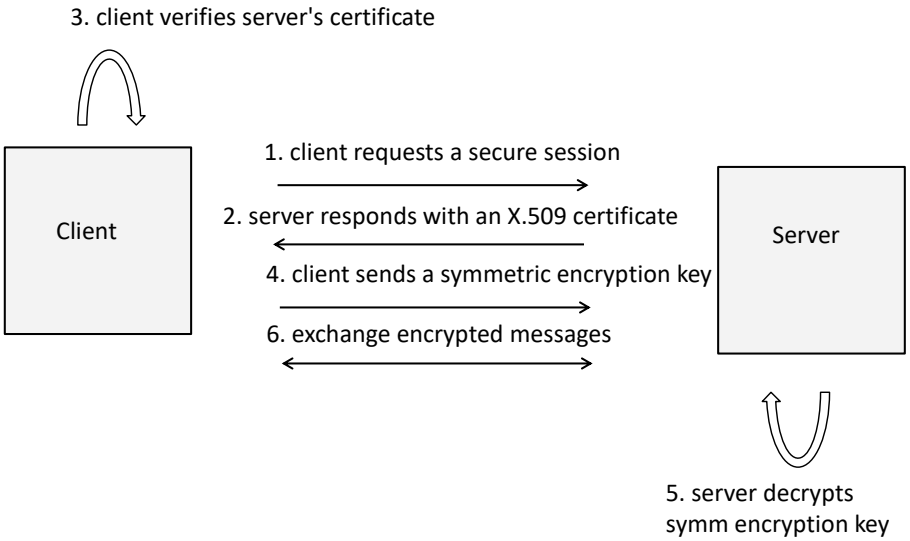
Symmetric Encryption



Asymetric Encryption



How SSL works



Identity & Credentials

- Identity is a digital entity
  - can be a user, group, organization, device, application
- Credentials are used as a mean of identification
  - Email, password, phone number

Authentication & Authorization

- Authentication (who is the user)
  - get the user's identity
  - different identification methods – username/password, fingerprint, other credential types
  - two-factor authentication – multiple factors
- Authorization (what is the user allowed to do)
  - categorize user to serveral roles

Authentication Modes

- Passive authentication
  - the user will be redirected to the Identity provider (IdP) to receive a signed token with claims
- Active authentication
  - client connects to the IdP, receives a token and uses the token to authenticate

Claim-based identity and authentication

- A claim carries some piece of information about the user
- A token contains one or more claims
- A token is created by a security token service (STS)

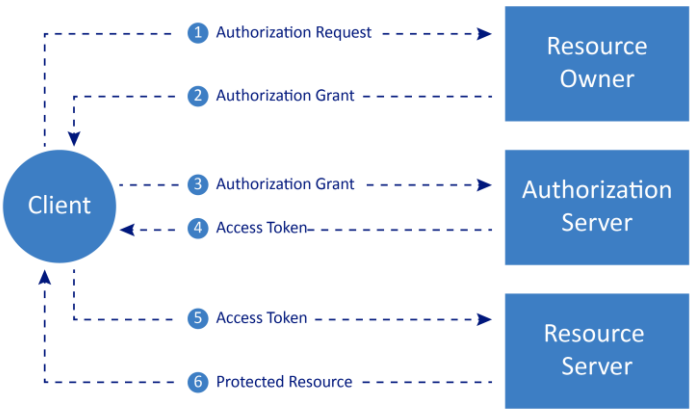
Claims-based authentication

1. The application requests a token from STS
2. The STS authenticates the user
3. The STS gets information about a user using the identity database or Active Directory
4. The STS creates and returns a token containing claims

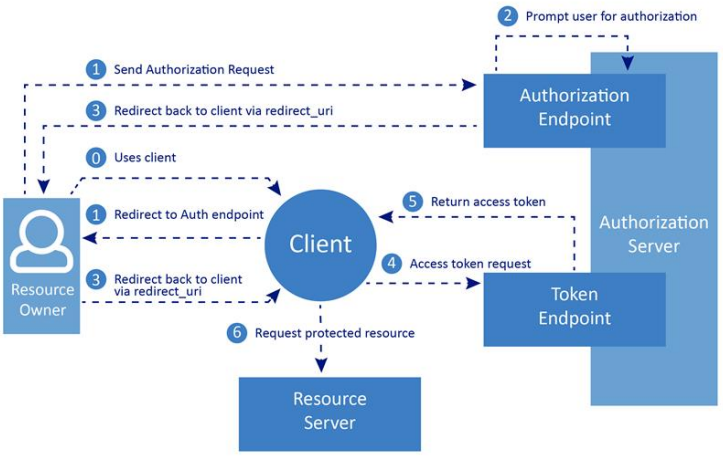
Claim-based identity flow

1. A browser or client gets a token from the STS provider
2. The client submits the token to the requested application
3. The application works with a set of trusted STS and verifies the token
4. Access is granted to the user based on the user's claims and roles

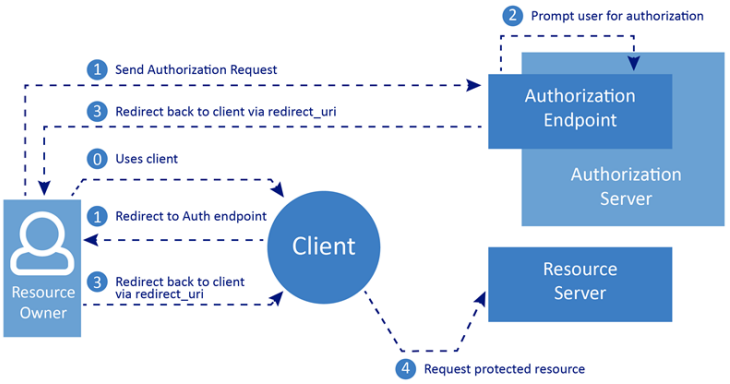
OAuth 2.0 abstract flow



OAuth 2.0 authorization code



OAuth 2.0 implicit flow



OpenID Connect 1.0

- Based on OAuth 2.0
- Adds identities and authentication
- Standard for applications to get user-information
- Supports authentication code and implicit flows

Azure AD

- Is an identity provider
- Is a directory service
- Exposes user data as tokens containing claims
- Supports groups and role management
- Supports identity federation
- Supports SAML and JWT tokens

OpenID Connect Middleware

- Microsoft.IdentityModel.Clients.ActiveDirectory
- Implements the AuthPropertiesTokenCache class
- Configures authentication to use Azure AD
- Authentication middleware, sign-in and sign-out handlers

Azure AD B2C

- User identity management solution
- An identity provider
- Supports social identity providers (Microsoft, Google, Facebook)
- Is an Azure AD with a specific application for B2C

Summary

- Authentication
- Authorization
- Identities

AZURE FUNCTIONS

Overview

- Serverless
- Triggers
  - HTTP
  - Timers
  - Cosmos, Blob, Queue, Event…
- Hosting
  - Consumption-based
  - App Service plan
- Automatic scaling

Consumption Plan Limits

- Execution Timeout
  - 5 minutes, can be extended to 10
- All HTTP Functions have a max time of 230 seconds (timeout Azure Load Balancer)

HTTP Trigger

```
[FunctionName("Function1")]
public static async Task<IActionResult> Run(
  [HttpTrigger(AuthorizationLevel.Function, "get", "post", Route = null)]
  HttpRequest req, ILogger log)
{
  log.LogInformation("C# HTTP trigger function processed a request.");

  //...
  return name != null
    ? (ActionResult)new OkObjectResult($"Hello, {name}")
    : new BadRequestObjectResult("Please pass a name on the query string
      or in the request body");
}
```

Functions with DI

- FunctionsStartup Attribute

```
public class Startup : FunctionsStartup
{
  public override void Configure(IFunctionsHostBuilder builder)
  {
    builder.Services.AddDbContext<BooksContext>(config =>
    {
      config.UseSqlServer(Environment.GetEnvironmentVariable(
        "BooksConnection"));
    });
  }
}
```

More Features

- Function Proxy
  - Single API, route requests to smaller function APIs
  - Modify requests and responses
- Premium Plan
  - Pre-warmed instances
  - Elastic scale out

Summary

FAAS

SCALABILITY

REAL TIME COMMUNICATION WITH SIGNALR AND WEBHOOKS

Bonus Chapter 3 – Professional C# 7

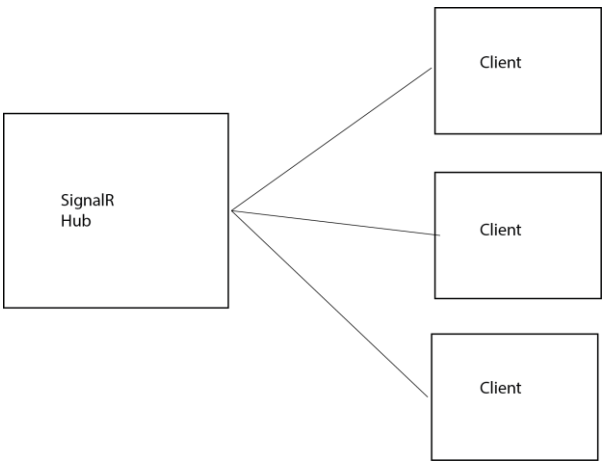
Overview SignalR

Real-time Communication

Uses WebSockets if possible

.NET, JavaScript, and other Libraries

SignalR Architecture



SignalR Hub

- Send Messages to Clients
  - All Clients
  - Group of Clients
  - A Specific Client

Client

Connects to a Hub

HubConnection

Create a Proxy

Send Messages to the Hub

Receive Messages

Grouping

Hub with IGroupClient

Clients can join and leave a group

Azure SignalR Service

Take load of client connections

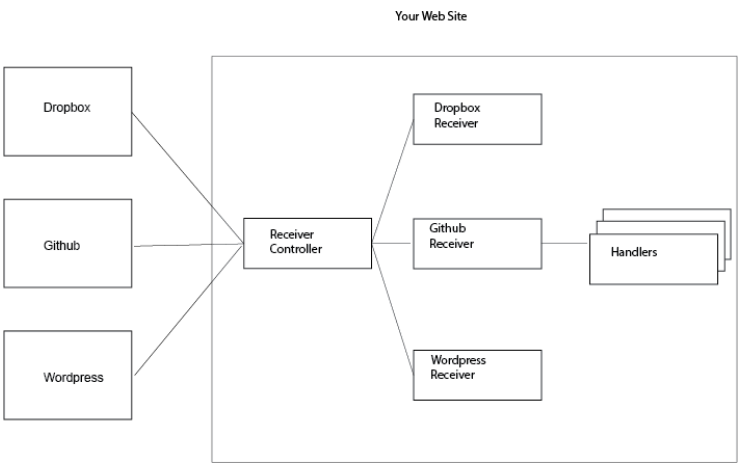
Can be integrated with ASP.NET Core

Can be used in a serverless app environment

WebHooks Overview

- An API Contract
- Server becomes a Client and Calls into your standard Service
- Many Providers
  - Dropbox, GitHub, WordPress, SalesForce, Slack, Paypal…

WebHooks Architecture



Summary

- SignalR and WebHooks can be used in combination
- WebHooks service must be Internet accessible
- ASP.NET Core offers an easy way to create WebHooks senders & receivers

GRPC & WORKERS

Workers

Run Background Process

Use Host class for DI, logging, configuration

Run as Windows Service

- Package Microsoft.Extensions.Hosting.WindowsService
- UseWindowsService

```
dotnet publish –o c:\code\workersample
sc create workersample binpath=c:\code\workersample\workersample.exe
```

gRPC

Binary communication between services

Protobuf wire transfer

Summary

Workers

gRPC

MICROSERVICES

https://devblogs.microsoft.com/cesardelatorre/free-ebookguide-on-net-microservices-architecture-for-containerized-net-applications/

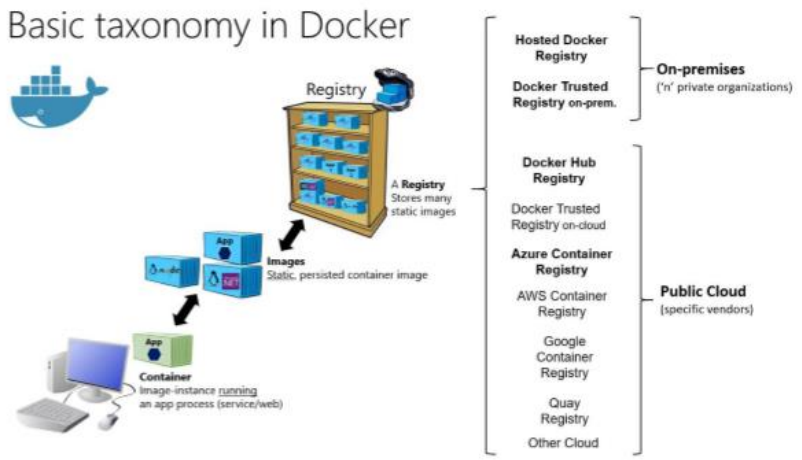
Overview

- Small in size
- Messaging-enabled
- Bounded by contexts
- Autonomously developed
- Independently deployable
- Decentralized
- Built and released with automated processes

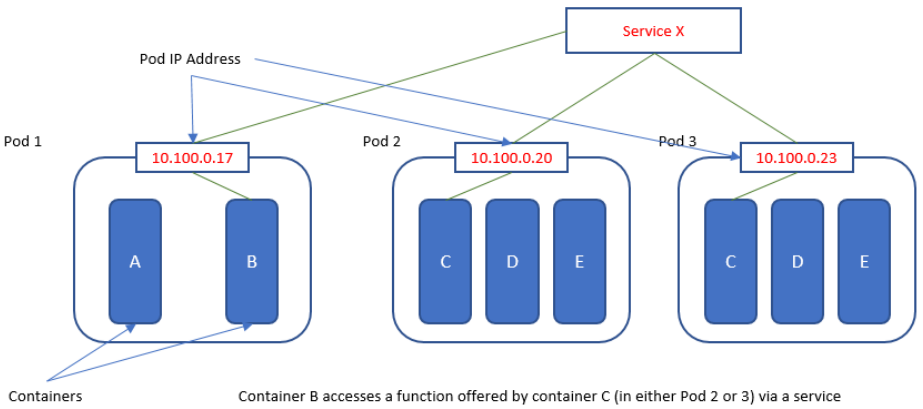
Docker

- Container image
  - package with dependencies and information needed to create a container
- Dockerfile
  - instructions to build a docker image
- Container
  - instance of a docker image
- Volumes
  - writeable filesystem managed by Docker
- Tag
  - mark or label apply to images
- Repository
  - collection of docker images
- Registry
  - service provides access to repository

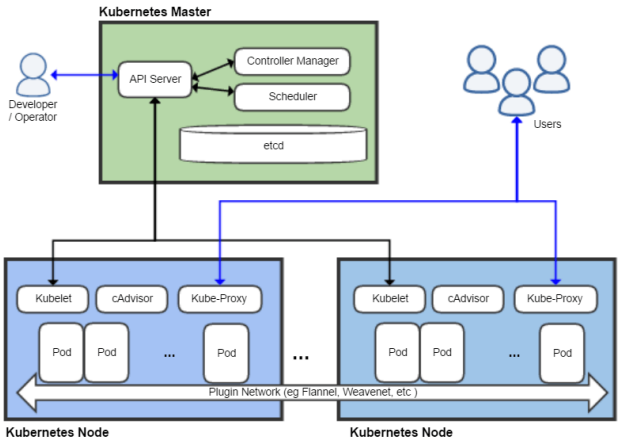
Taxonomy in Docker



Run Containers with Kubernetes Clusters



Kubernetes Cluster and Nodes



Kubernetes Terms

- Kubernetes Control plane
  - Main controlling unit of the cluster
  - etcd
    - key-value store with configuration
  - API Server
    - REST Server, configure workloads
  - Scheduler
    - Schedule nodes to run on
  - Controller manager
    - drives cluster state
    - create, update, delete resources (pods, endpoints..)
- Kubernetes node
  - known as Worker or Minion
  - Workloads are deployed
  - Kubelet
    - run state on each node
  - Kube-proxy
    - network proxy and load balancer
  - Container runtime
    - resides inside a pod
    - supports Docker container

ASP.NET Core Health Checks

- IHealthCheck
- AddHealthChecks
  - Add interface to container
- UseHealthChecks
  - Configure middleware

Summary

MICROSERVICES

DOCKER

KUBERNETES