Трудности перехода на ASP.Net Core и Docker

Александр Иванов
alexander.ivanov@arcadia.spb.ru

План

1. Что было
2. Что хотелось
3. Что потеряли и нашли
4. Работа и отладка
5. Сборка и развёртывание
6. Выводы

Что было. Структура

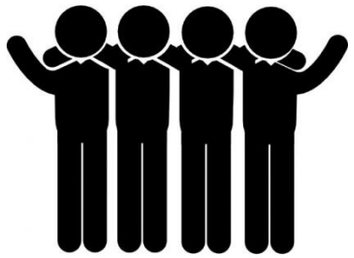
Microsoft® ASP.net™

WebForms

ASP.net MVC

Microsoft® .NET4.5™

C++



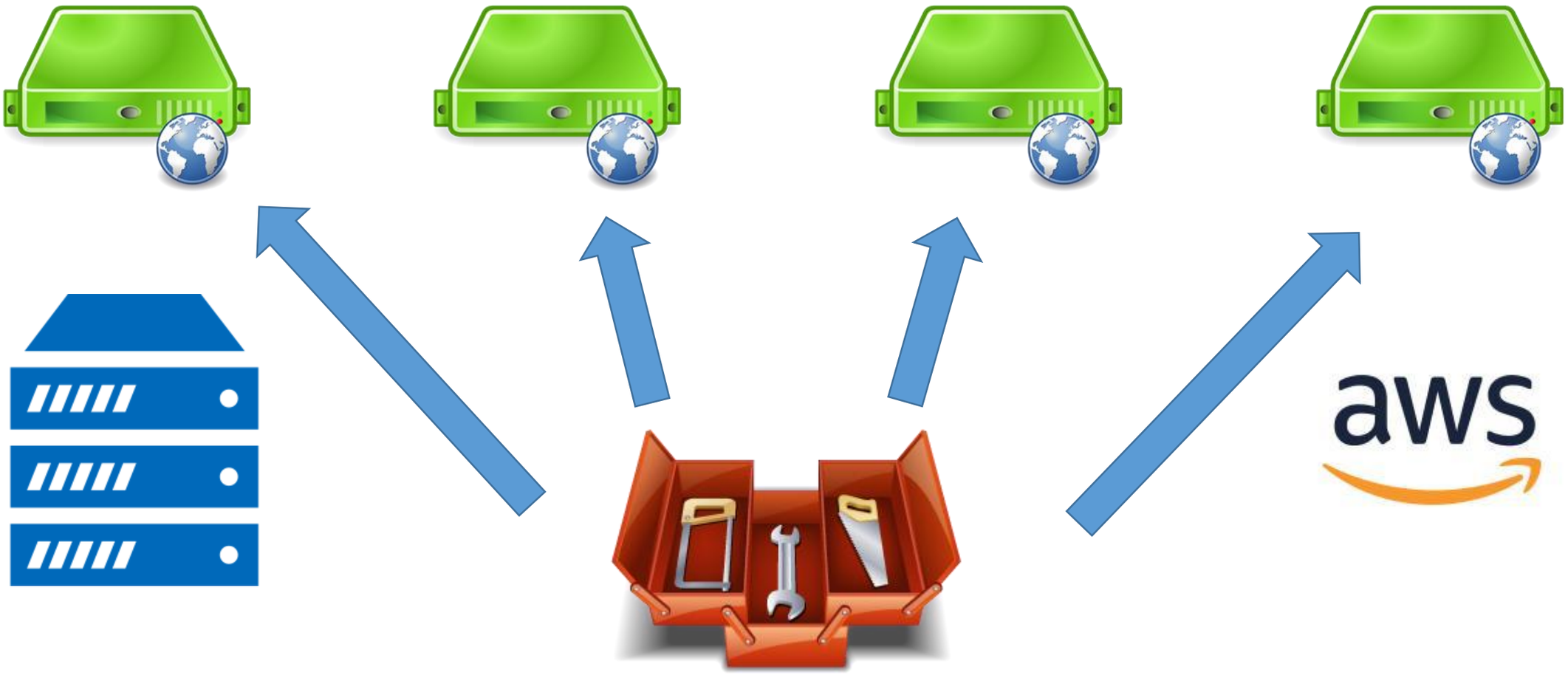
x25

App.csproj

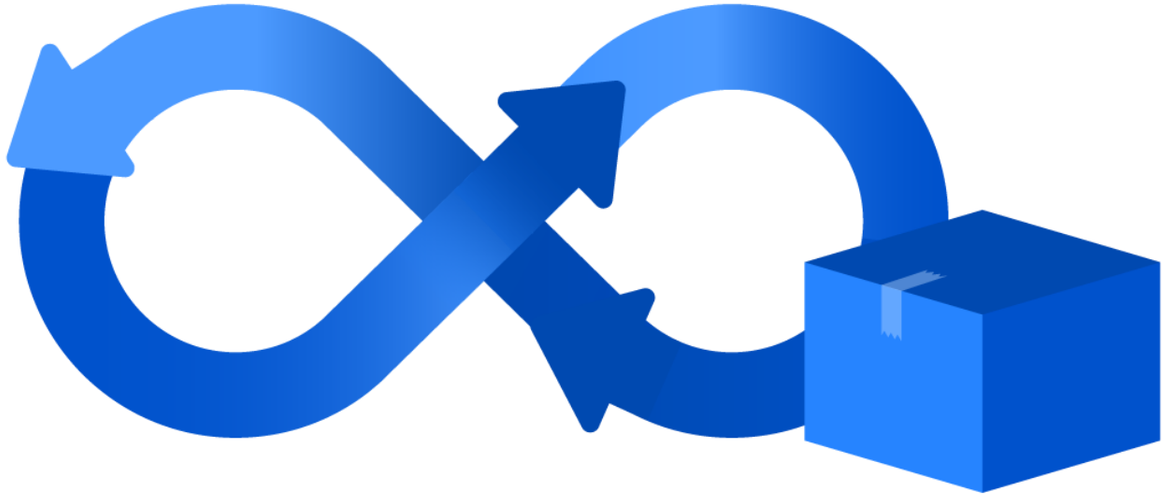
>500

VBSCRIPT

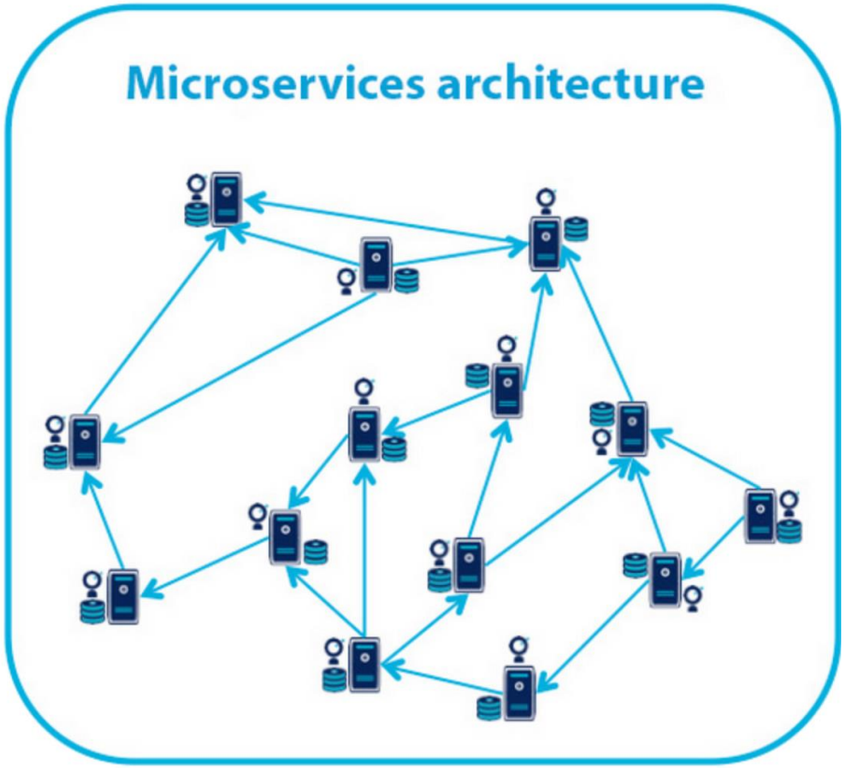
Что было. Развёртывание



Направление



Continuous Deployment

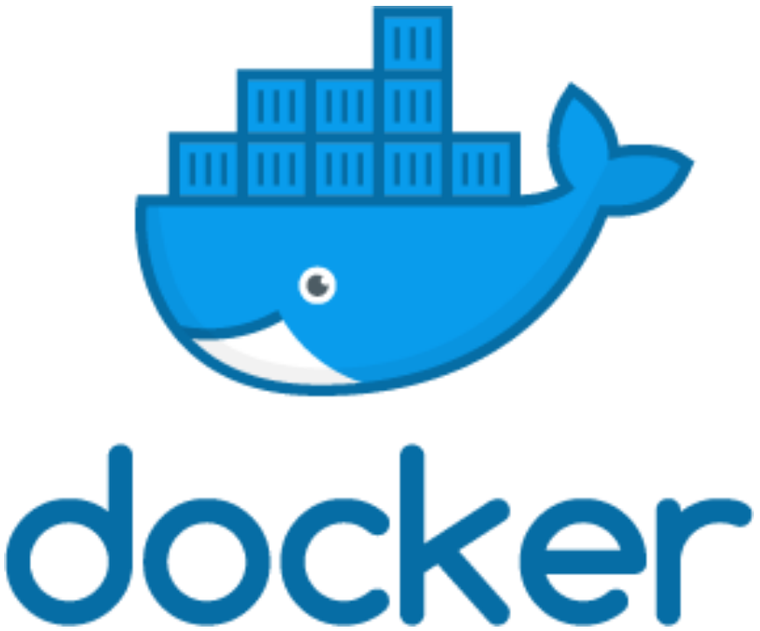


Microservices

Что хотелось. Заказчику

- Новые приложения
- RestApi, WCF
- RabbitMQ
- Stateless
- Amazon Web Services
- Масштабируемые
- Защищённые

Что хотелось. Команде



Почему ASP.Net Core?

- Почему бы и нет?
- The Twelve-Factor App
- Nuget
- 1.1 и 2.0 > 1.0
- .Net Standard 2.0
- Сообщество
- FAQ
- Кросс-платформенность
- Дешевле в содержании

Почему Docker?

- AWS Container Service

- Скорость развёртывания
- Простота
- Изолированность
- Масштабируемость
- ...
- Дешевизна

Что хотелось. Мнения

Команда

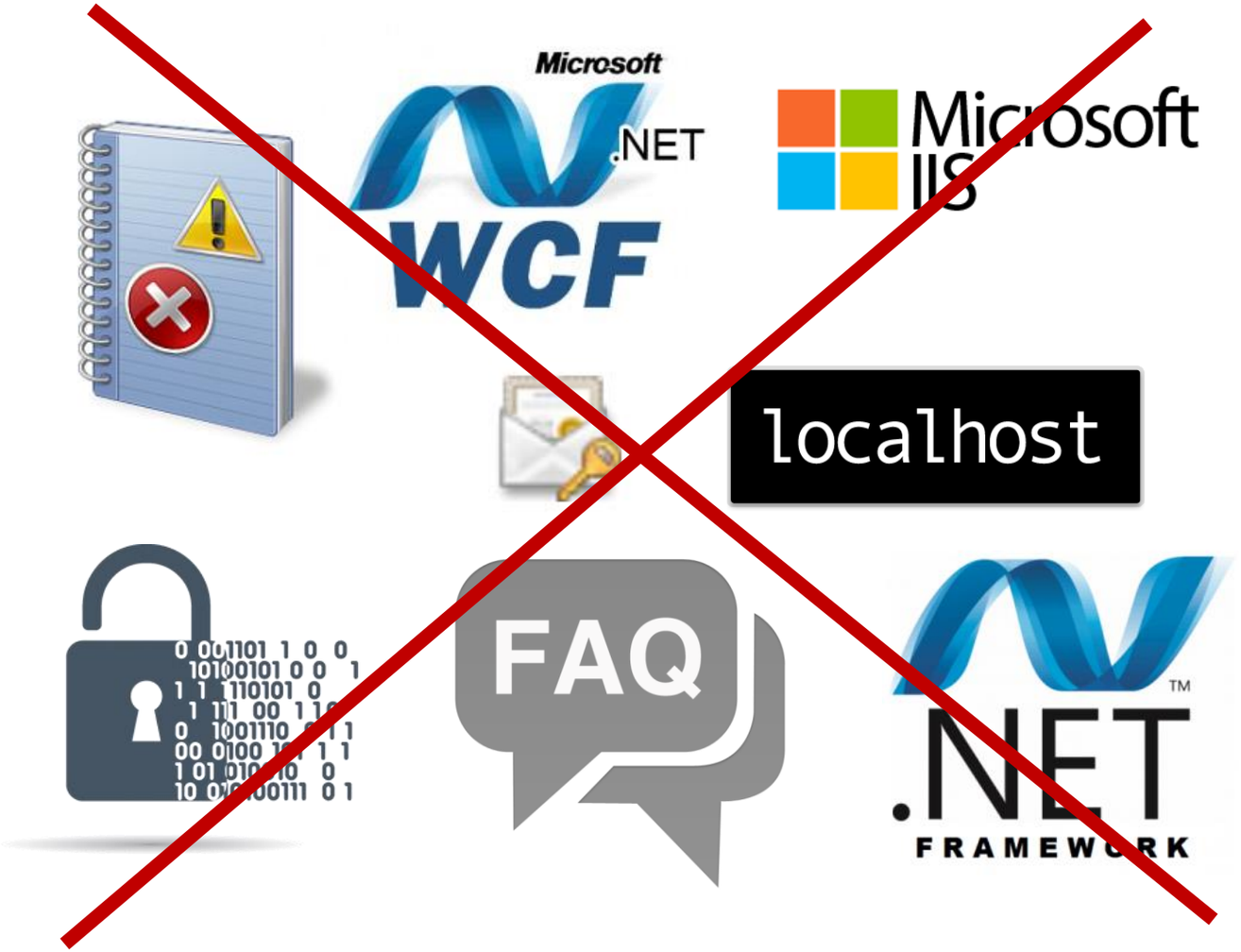
Ops

Архитекторы

Менеджеры

Команда

Что мы потеряли



~~Event log~~

Стандартный логгер 😐

Serilog! Просто, быстро, настраиваемо

```
{Timestamp:yyyy-MM-dd HH:mm:ss} [{Level}]
{Message} {Properties}{NewLine}

"Enrich": ["FromLogContext"]

using (LogContext.PushProperty("Prop", value))
```

Certificate Store для самозаверенных сертификатов

```
if (!env.IsProduction())
{
    var clientHandler = new HttpClientHandler();

    clientHandler.ServerCertificateCustomValidationCallback += (…) =>
    {
        X509Chain chain2 = new X509Chain();

        chain2.ChainPolicy.ExtraStore.Add(new X509Certificate2(RootCertificate));

        chain2.Build(cert);

        if (chain2.ChainStatus.Length == 0)
            return true;

        return chain2.ChainStatus[0].Status == X509ChainStatusFlags.NoError
            || chain2.ChainStatus[0].Status == X509ChainStatusFlags.UntrustedRoot;
    };
}
```

~~Полная поддержка WCF~~

- Поддерживаемые функции WCF

- 'TransportWithMessageCredential' is not supported

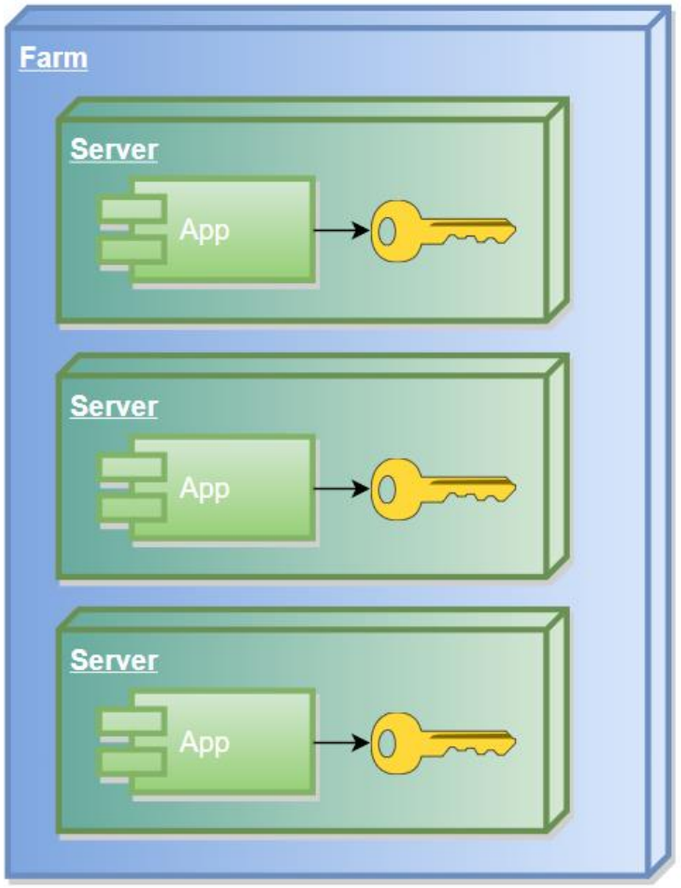
- SimpleSOAPClient

```
var requestBody = DataSerializer.ObjectToXmlDataString(soapBody);

var envelope = SoapEnvelope
    .Prepare()
    .Body(XElement.Parse(requestBody))
    .WithHeaders(
        KnownHeader.Oasis.Security.UsernameTokenAndPasswordText("user","pass"));
```

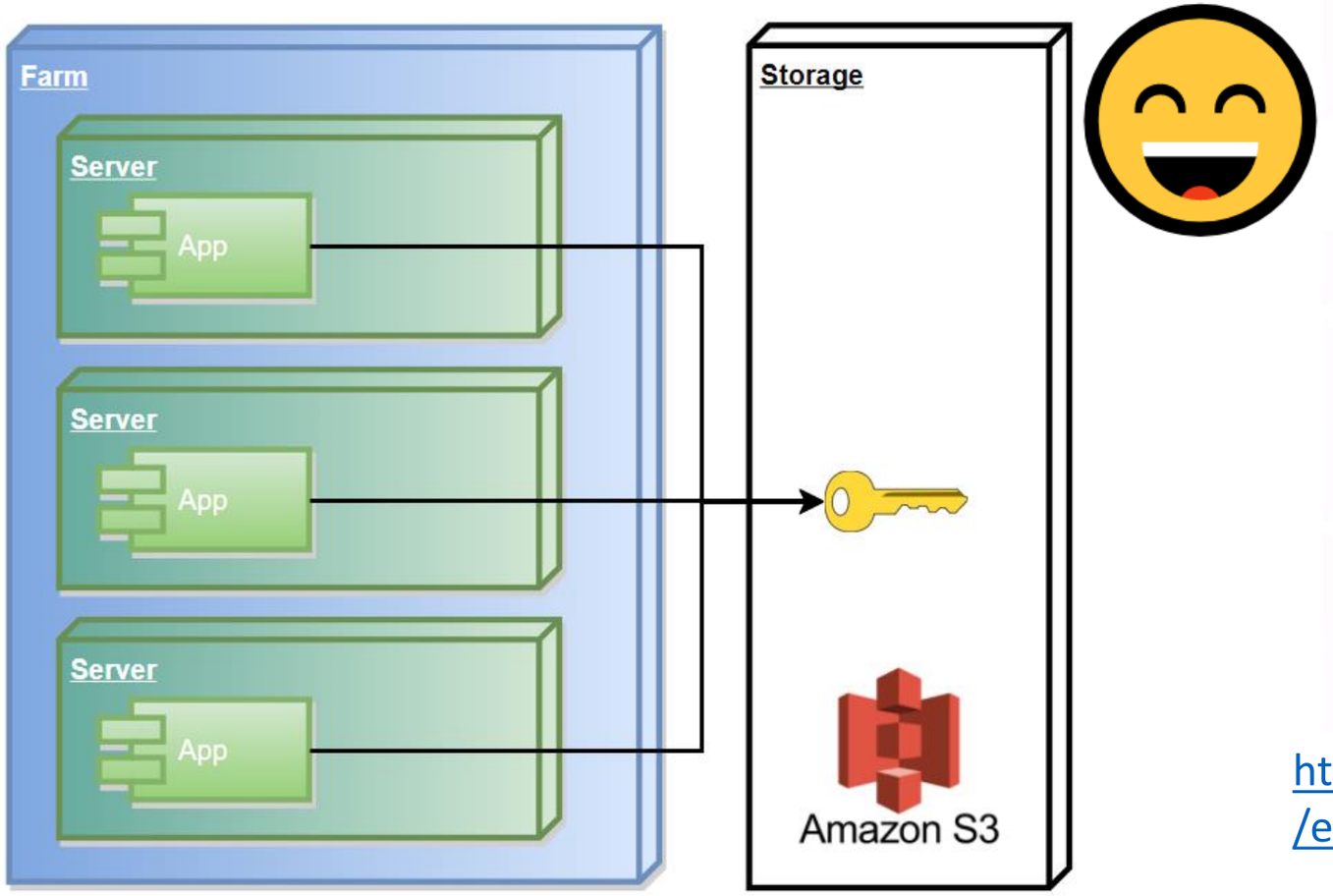
~~IIS и развёртывание веб-сайтов на нём~~

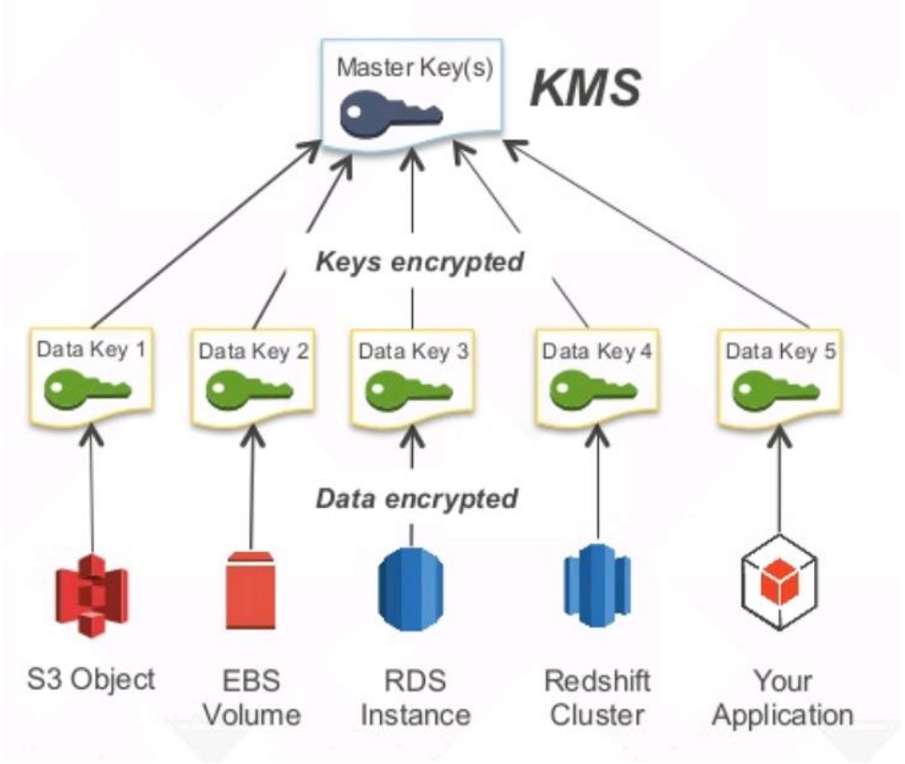
- 
- Kestrel
- Elastic Load Balancing
- SSL termination
- Amazon Elastic Container Service

~~Machinekey и validationkey для шифрования~~



~~Machinekey и validationkey для шифрования~~





https://www.slideshare.net/AmazonWebServices/encryption-and-key-management-in-aws

```
services.AddDataProtection().PersistKeysToAwsS3(…).ProtectKeysWithAwsKms(…);
```

~~Basic authentication~~

Правда надо?
Легко

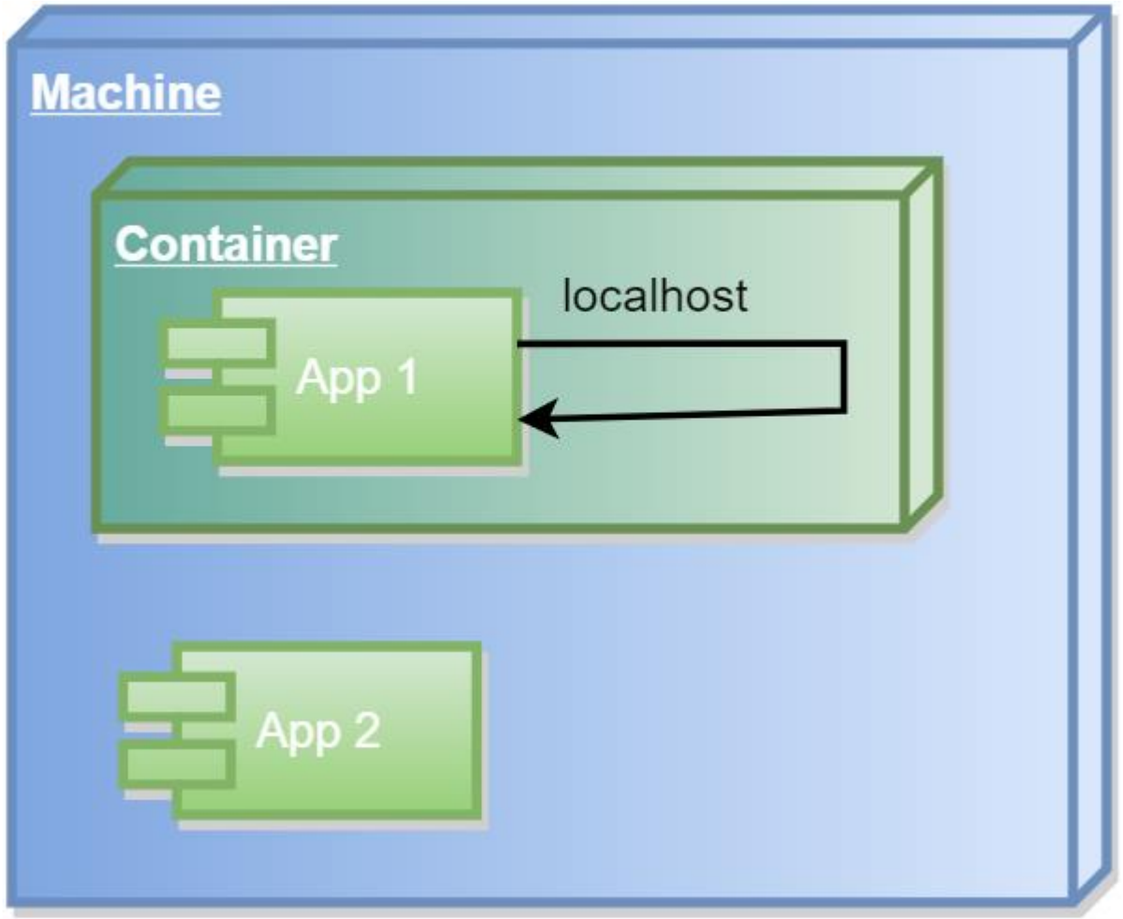
https://github.com/blowdart/idunno.Authentication

https://github.com/msmolka/ZNetCS.AspNetCore.Authentication.Basic

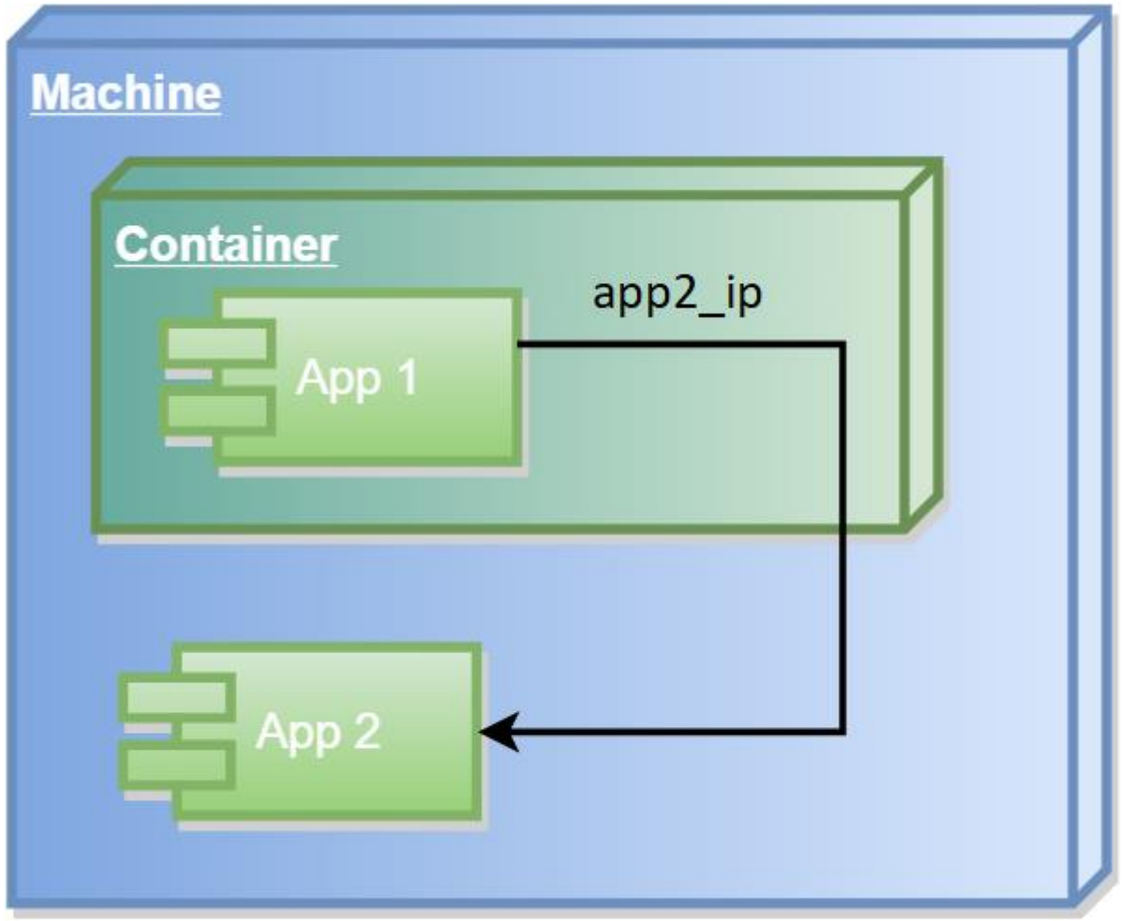
https://github.com/edjCase/BasicAuth

```
services.Configure<BasicAuthenticationOptions>("Basic", …);

services.AddAuthentication().AddBasicAuthentication();

app.UseAuthentication();
```

~~localhost~~



~~localhost~~




```
/etc/hosts
app2_ip                 10.0.0.10
```

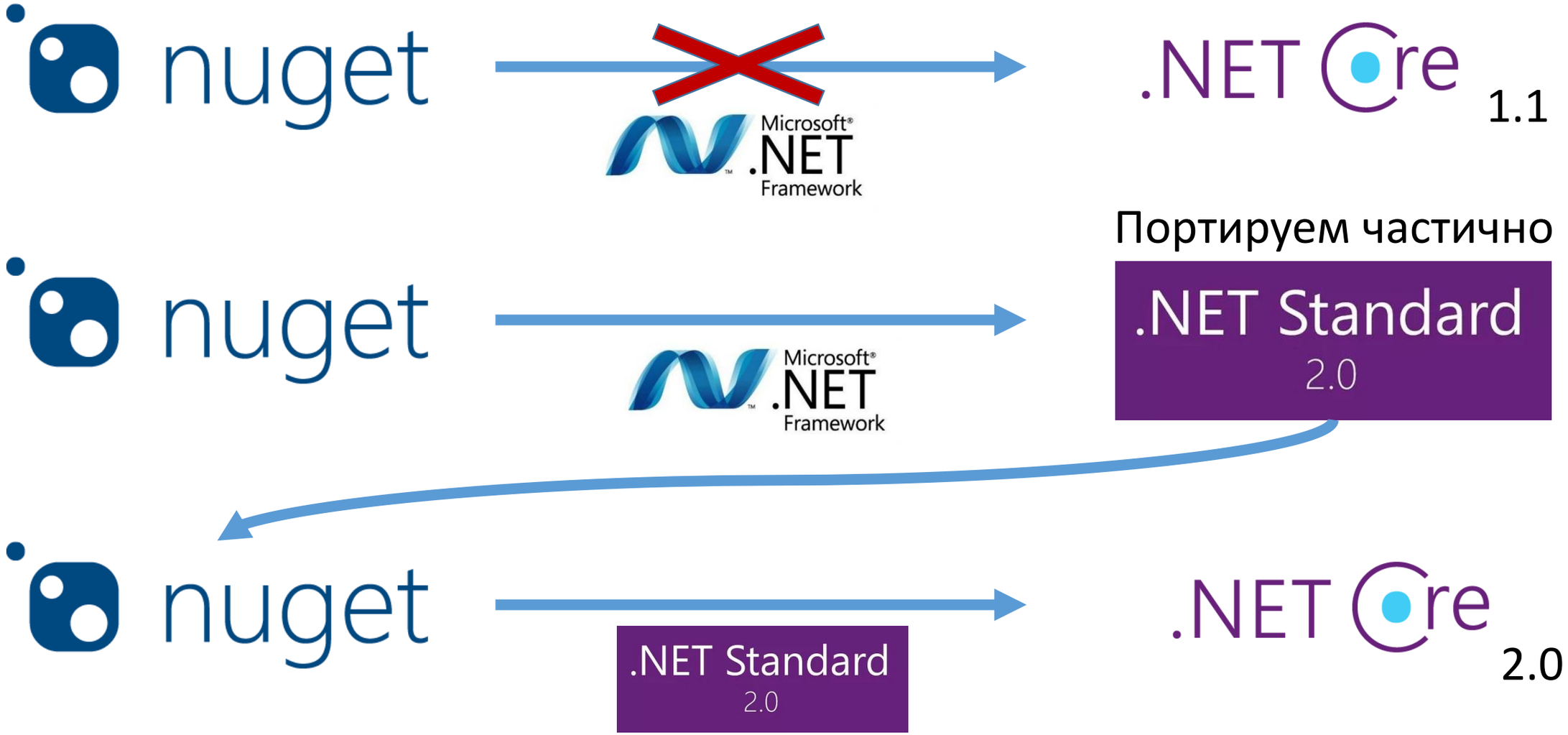
docker run -it
--add-host app2_ip:10.0.0.10

```
extra_hosts:
    - "app2_ip:10.0.0.10"
```

```
extra_hosts:
    - "app2_ip:${APP2_IP}"
```

```
.env
APP2_IP=10.0.0.10
```

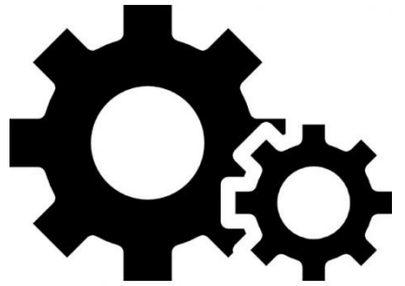
~~Собственные наработки и библиотеки, написаные под .Net Framework~~



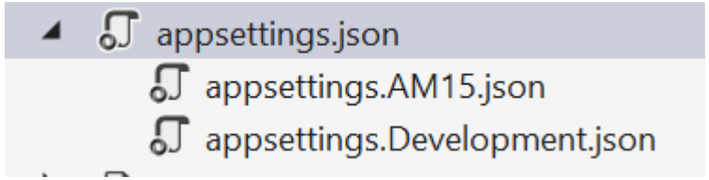
~~Быстрый поиск ответов и статичность подходов~~

- open-source
- Версии
  - .Net Core, Visual Studio, Docker, Docker Compose
- Ответы
  - .Net Core 1.0, .Net Core 1.1, .Net Core 2.0, .Net Framework
- Отладка, сборка и развёртывание
- Танцы с бубнами

Что мы нашли



Гибкость настройки



+ Переменные среды

```
var builder = new ConfigurationBuilder()
    .SetBasePath(env.ContentRootPath)

    .AddJsonFile("appsettings.json", optional: false)

    .AddJsonFile($"appsettings.{env.EnvironmentName}.json", optional: true)

    .AddEnvironmentVariables();


WebHost.CreateDefaultBuilder(args).…
```

Гибкость настройки

```
services.AddOptions();

services.Configure<ApplicationSettingsProvider>(configuration);

services.Configure<SpecificSettings>(configuration.GetSection("SpecificSection"));

public ApplicationClass(IOptions<ApplicationSettingsProvider> settings)
{
    var setting = settings.Value.MySetting;
}

public ApplicationClass(IOptionsSnapshot<SpecificSettings> settings)
{
    var setting = settings.Value.MySetting;
}
```

https://docs.microsoft.com/en-US/aspnet/core/fundamentals/configuration/options

Гибкость настройки

```
var options = configuration.GetAWSOptions();

options.Credentials = new BasicAWSCredentials(amazonKeyId, amazonKeySecret);

services.AddDefaultAWSOptions(options);

services.AddAWSService<IAmazonS3>();
```

https://github.com/aws/aws-sdk-net

Конфигурация объектом

```
// MySettings.cs
public class MySettings
{
    public string MySetting { get; set; }
}

// ApplicationClass.cs
public ApplicationClass(IOptions<MySettings> settings)
{
    var setting = settings.Value.MySetting;
}

// Startup.cs
services.AddOptions();

services.Configure<MySettings>(Configuration);
```

Конфигурация интерфейсом

```
// AppSettingsProvider.cs
public class AppSettingsProvider : ISettingsProvider
{
    public string MySetting { get; set; }
}
// OldApplicationClass.cs
public class OldApplicationClass(ISettingsProvider settingsProvider)
{
    var setting = settingsProvider.MySetting;
}
// Startup.cs
services.Configure<AppSettingsProvider>(Configuration);

services.AddSingleton<ISettingsProvider>(sp =>
       sp.GetRequiredService<IOptions<AppSettingsProvider>>().Value);
```

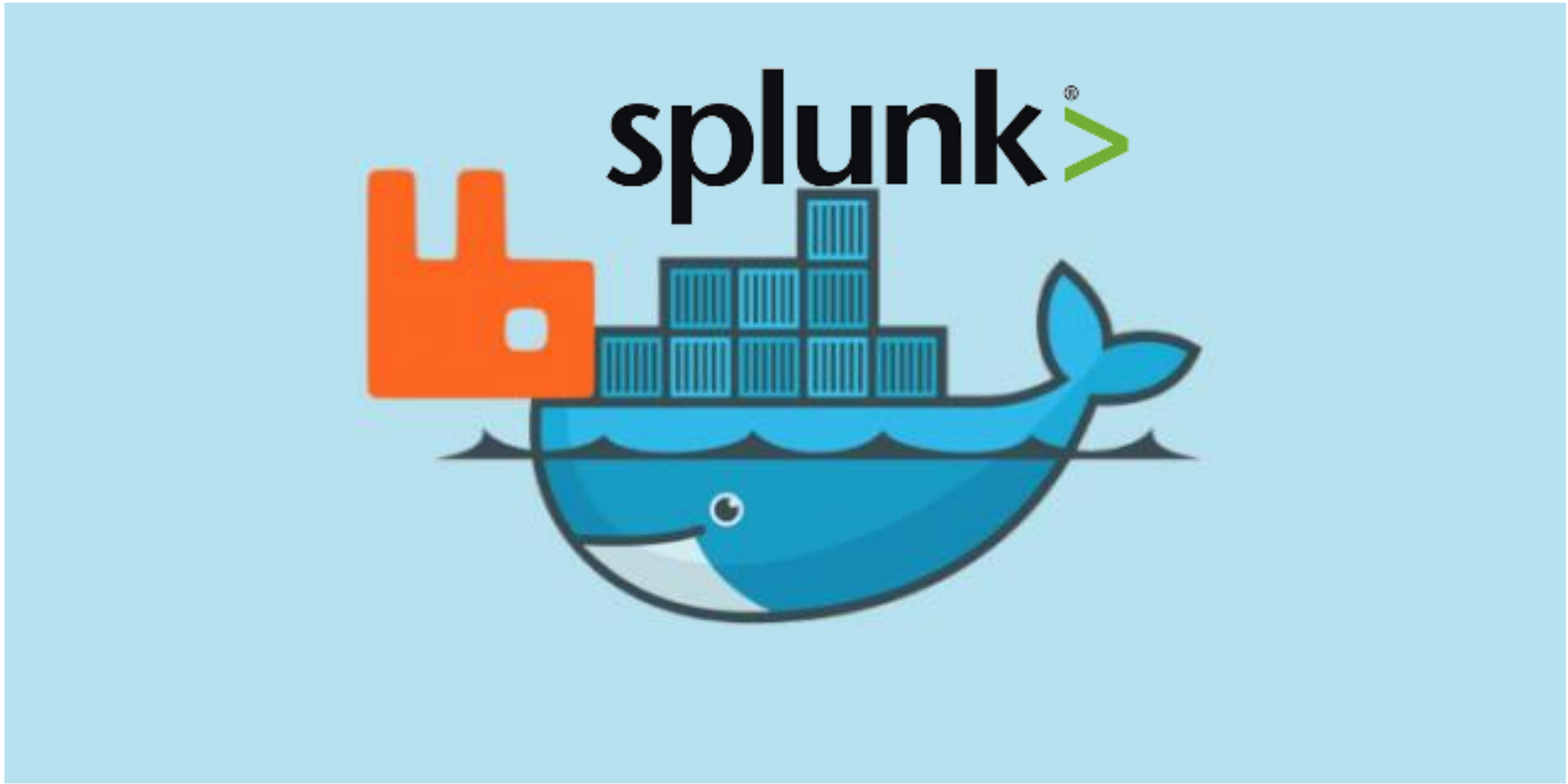
HttpContext

```
// Startup.cs
services.TryAddSingleton<IHttpContextAccessor, HttpContextAccessor>();


// ApplicationClass.cs
private readonly IHttpContextAccessor _httpContextAccessor;

public ApplicationClass(IHttpContextAccessor httpContextAccessor)
{
        _httpContextAccessor = httpContextAccessor;
}

var user = _httpContextAccessor.HttpContext.User;
```

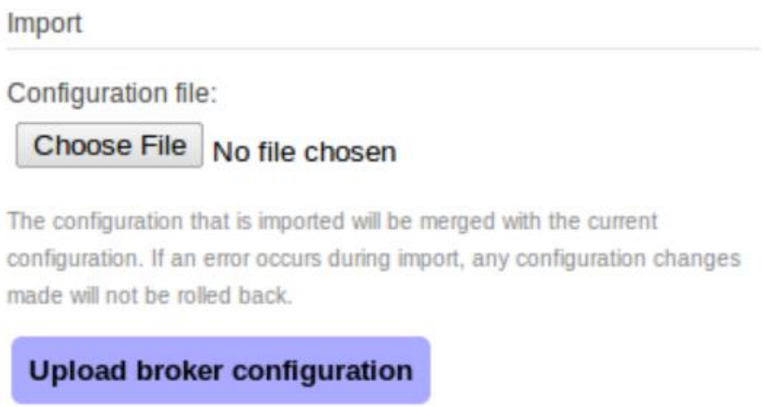
MVC + WebApi

```
DependencyResolver.SetResolver(new UnityDependencyResolver(container));
GlobalConfiguration.Configuration.DependencyResolver = new
Unity.WebApi.UnityDependencyResolver(container);

GlobalFilters.Filters.Add(new ExceptionFilter());
GlobalConfiguration.Configuration.Filters.Add(new WebApiHandleExceptionAttribute());

RouteTable.Routes.MapMvcAttributeRoutes(constraintsResolver);

[System.Web.Mvc.RouteArea("myarea")]
[System.Web.Mvc.RoutePrefix("mycontroller")]
[System.Web.Mvc.Route("myaction")]

GlobalConfiguration.Configuration.MapHttpAttributeRoutes();
[System.Web.Http.RoutePrefix("myapiprefix/mycontroller")]
[System.Web.Http.Route("myaction")]
```

Инфраструктура в контейнерах



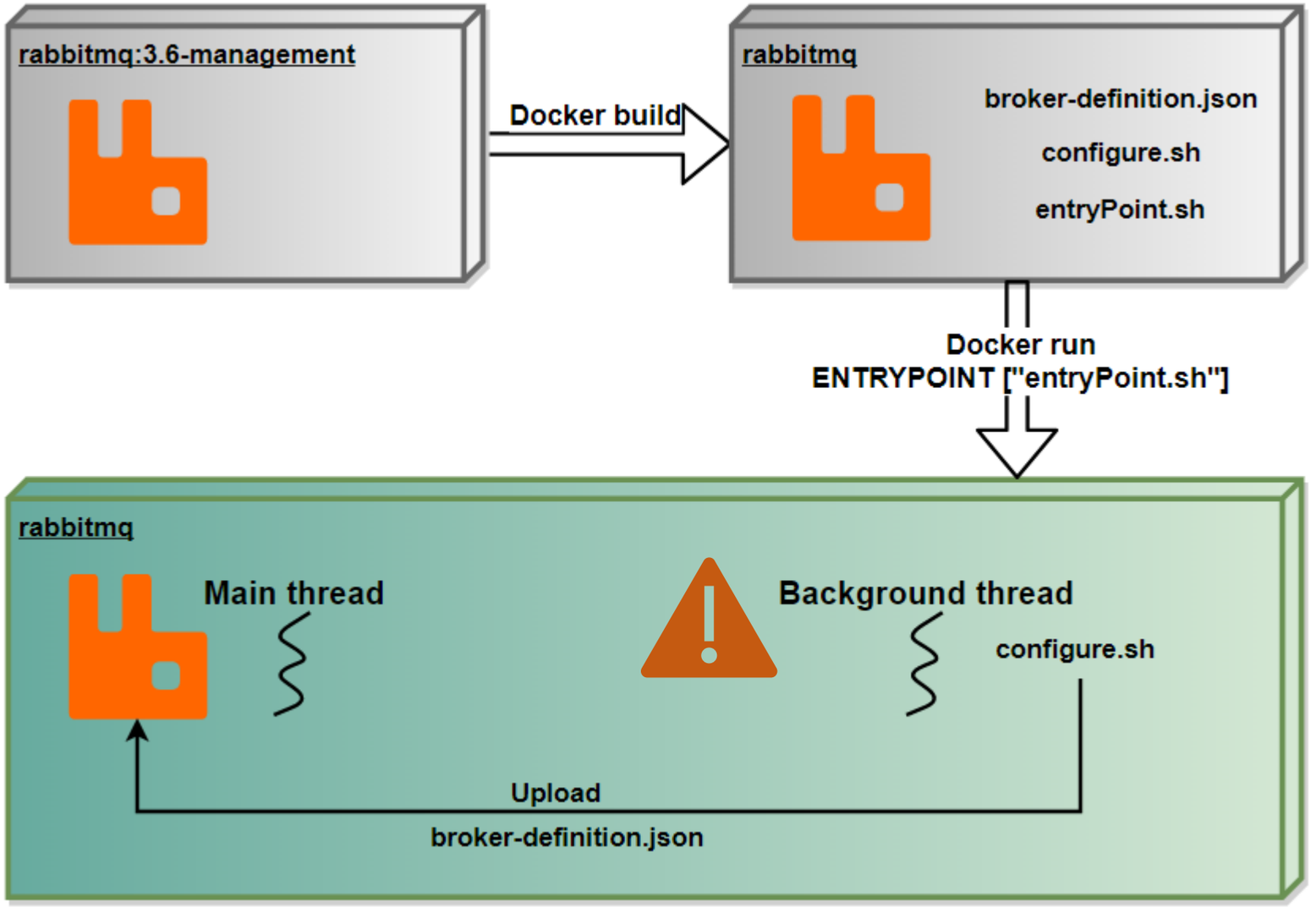
Настраиваем RabbitMQ. Конфигурация



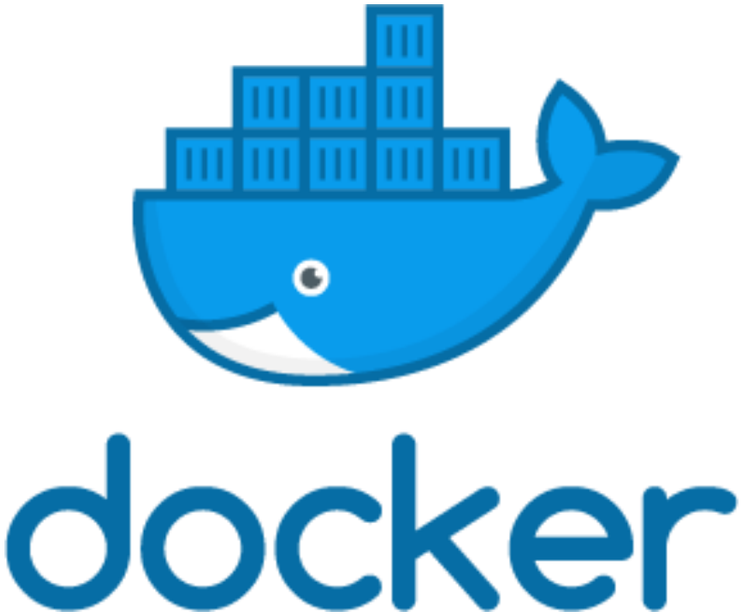
broker_definition.json

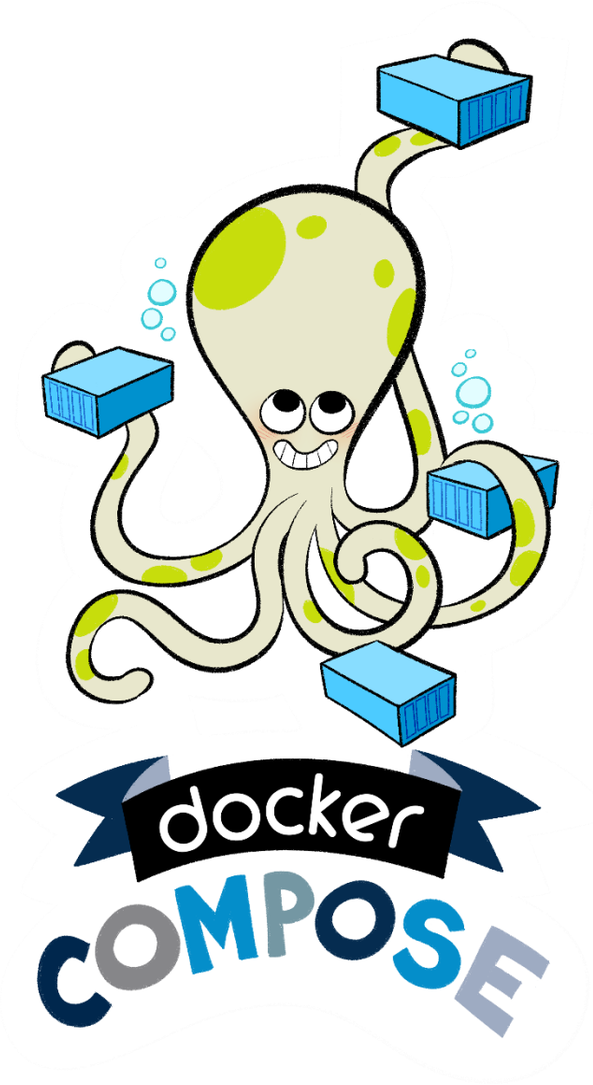
```
{
  "vhosts": [
    …
  ],
"exchanges": [
      …
  ],
 "queues": [
      …
  ],
 "bindings": [
      …
  ]
}
```

Настраиваем RabbitMQ. Автоматизация



Docker + Docker Compose

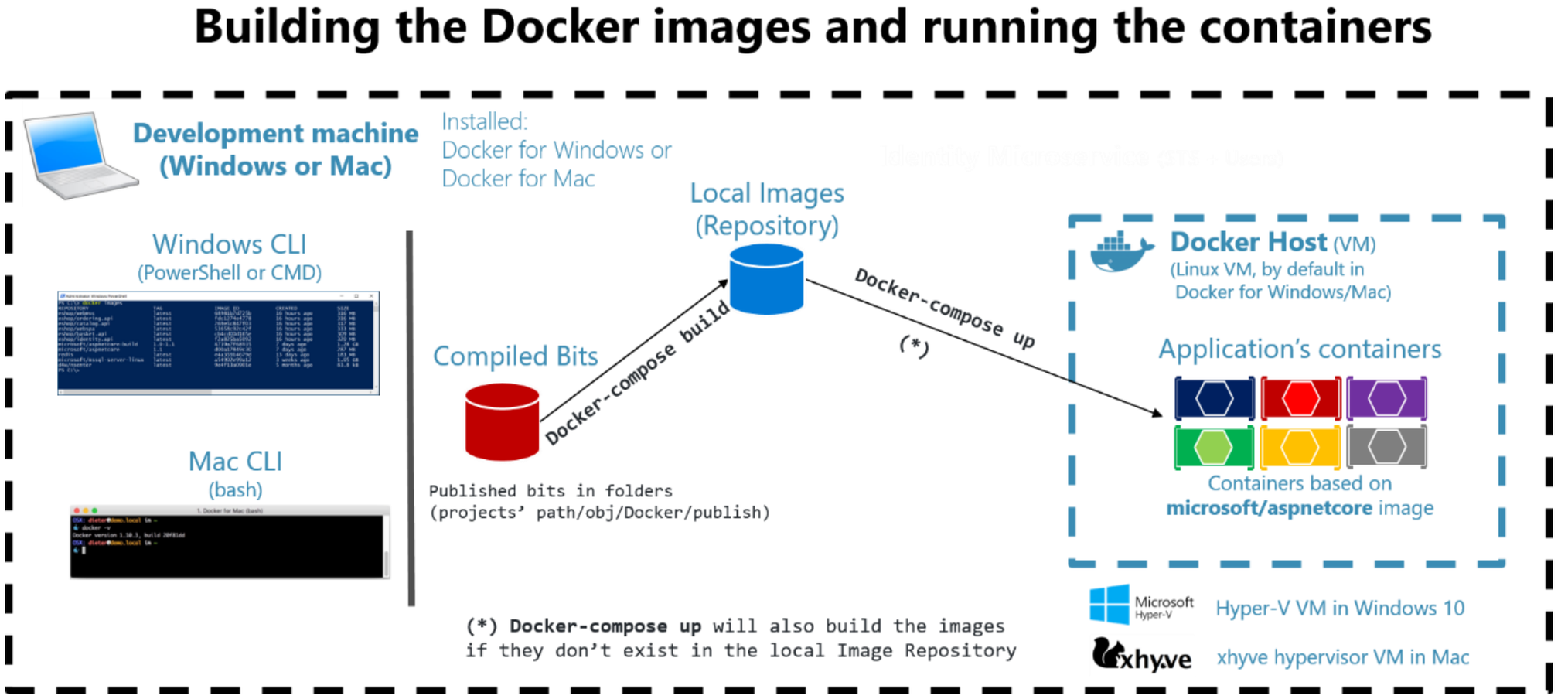




Dockerfile

```
FROM microsoft/aspnetcore:2.0
ARG source
WORKDIR /app
EXPOSE 80
COPY ${source:-obj/Docker/publish} .
ENTRYPOINT ["dotnet", "Application.dll"]
```

Docker Compose



https://docs.microsoft.com/en-us/dotnet/standard/microservices-architecture/multi-container-microservice-net-applications/multi-container-applications-docker-compose

Docker Compose. Синтаксис

```
services:
  myapp:
    image: myapp
    volumes:
      - /C/data:/files
    links:
      - myrabbitmq
    depends_on:
      - myrabbitmq
    ports:
      - "1000:80"
    extra_hosts:
      - "app2_ip:${APP2_IP}"
    build:
      context: ../Application
      dockerfile: Dockerfile
```

```
  myrabbitmq:
    image: myrabbitmq
    build:
      context: ../RabbitMQ
      dockerfile: Dockerfile
```

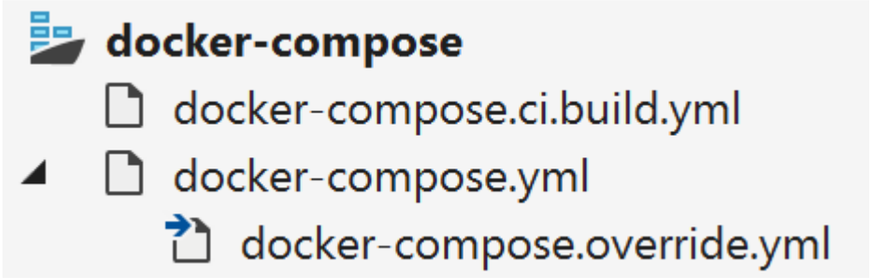
Docker Compose & Visual Studio 2017

docker-compose.yml

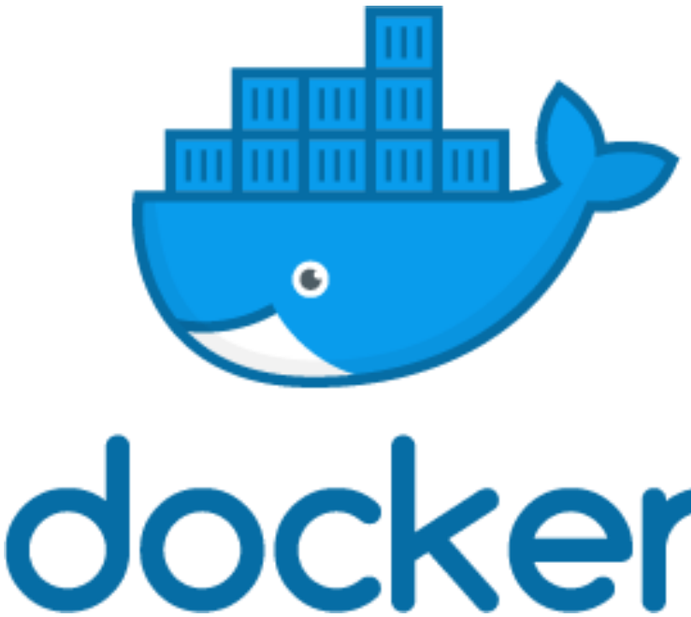
```
services:
  myapp:
    image: myapp
    build:
      context: ../Application
      dockerfile: Dockerfile
```

docker-compose.override.yml

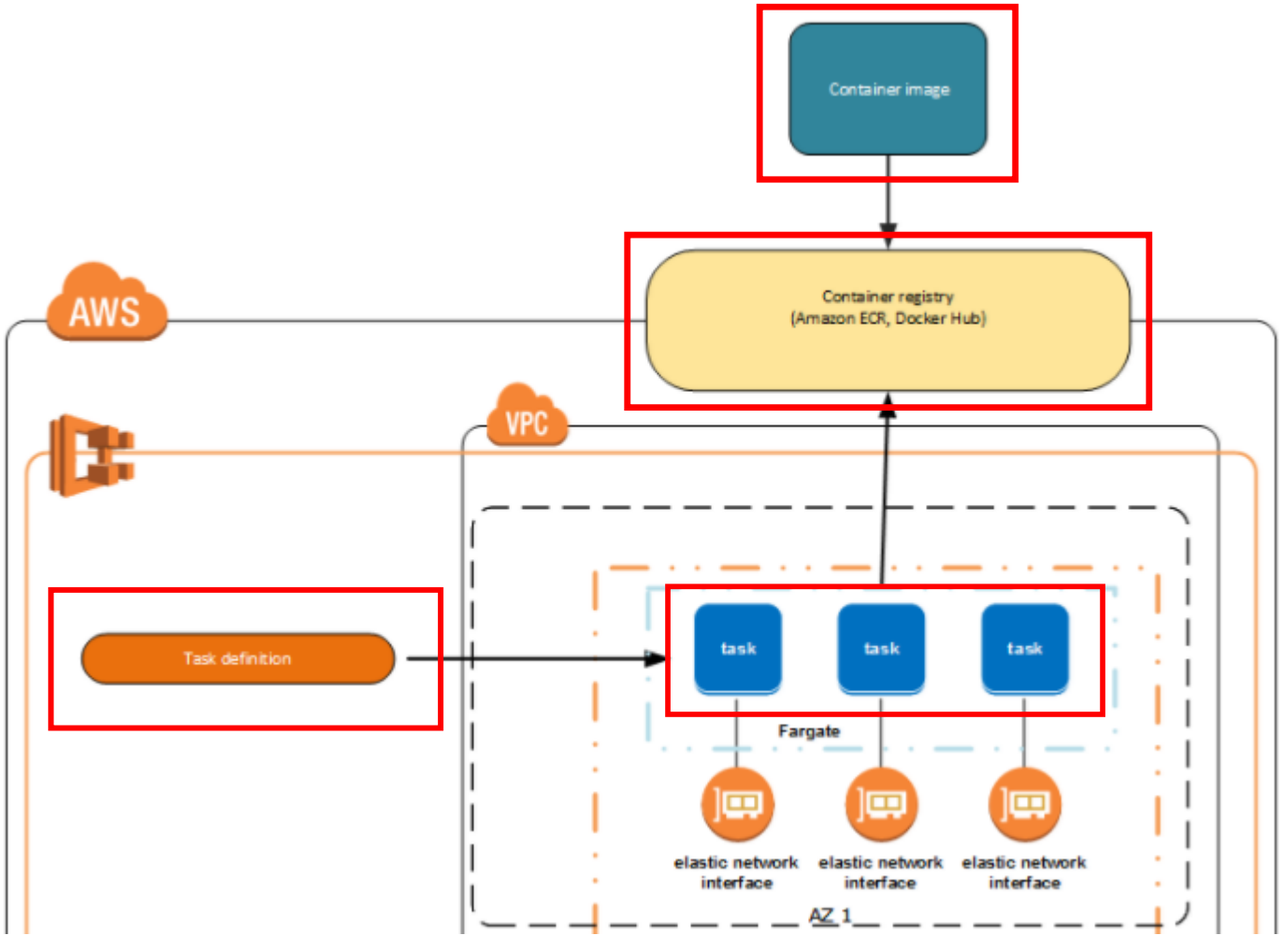
```
services:
  myapp:
    extra_hosts:
      - "app2_ip:${APP2_IP}"
    environment:
      - ASPNETCORE_ENVIRONMENT=Development
    ports:
      - "1000:80"
```

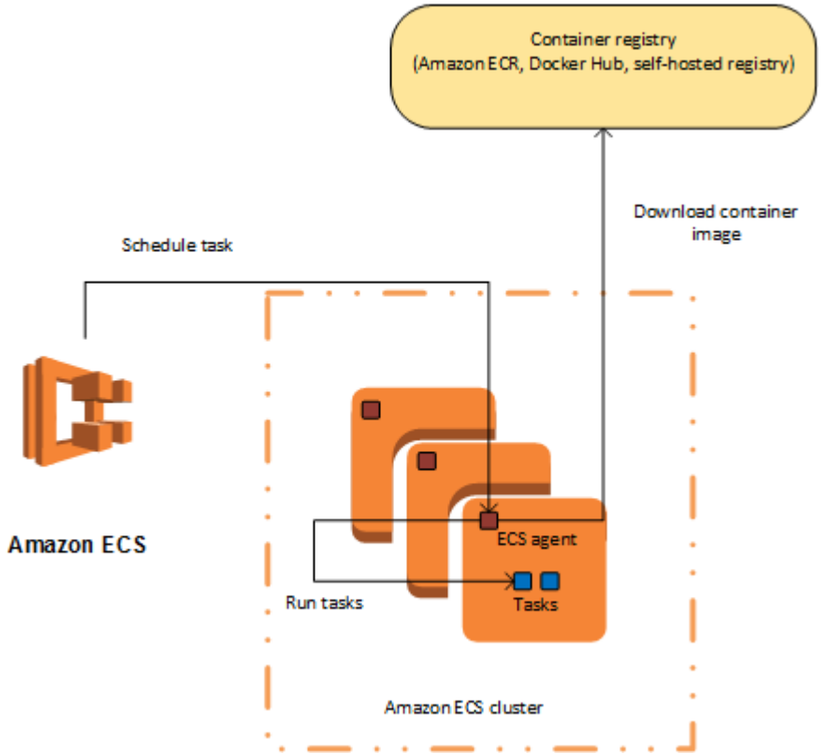


AWS Elastic Container Services



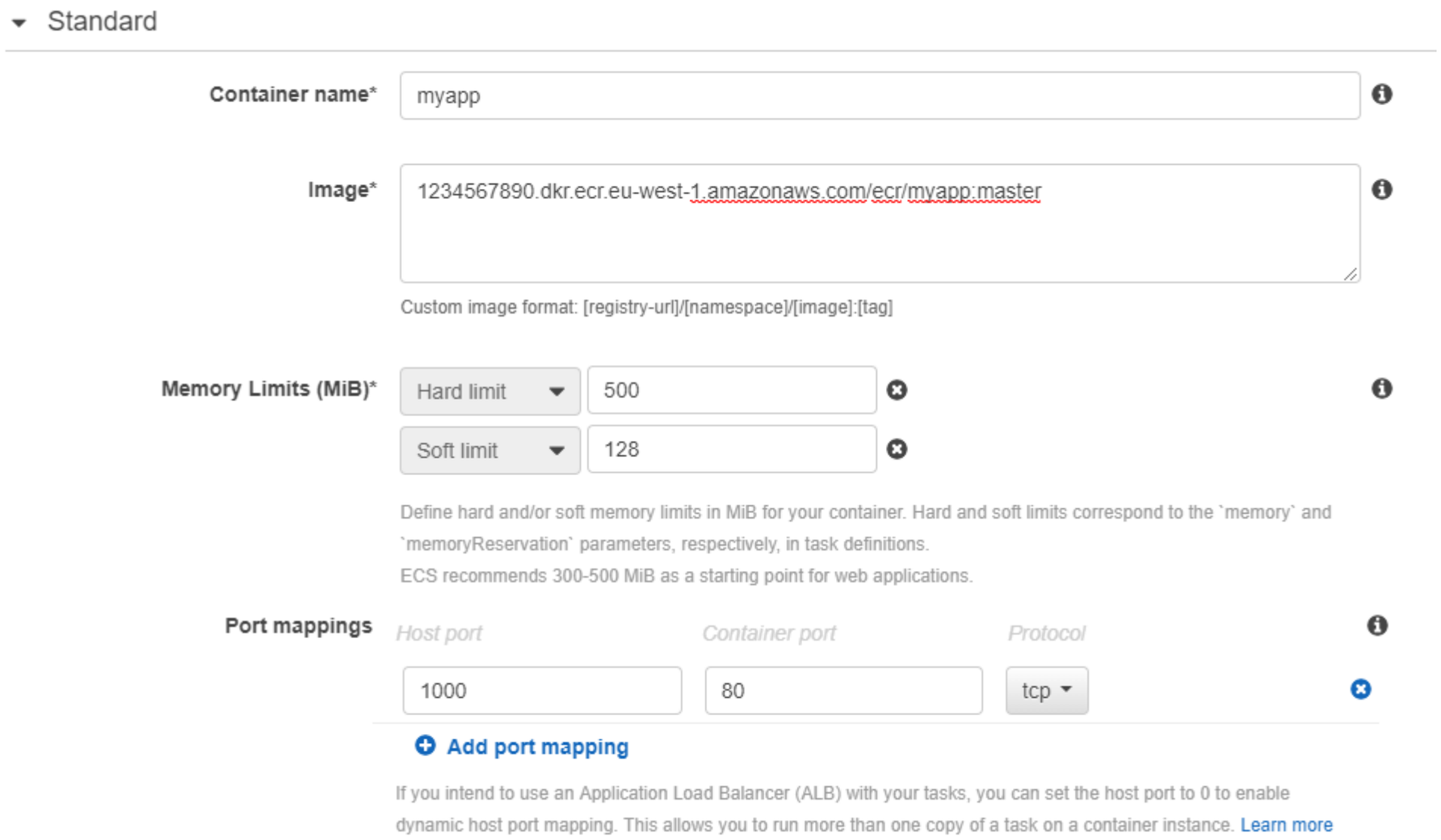
ECS. Схема



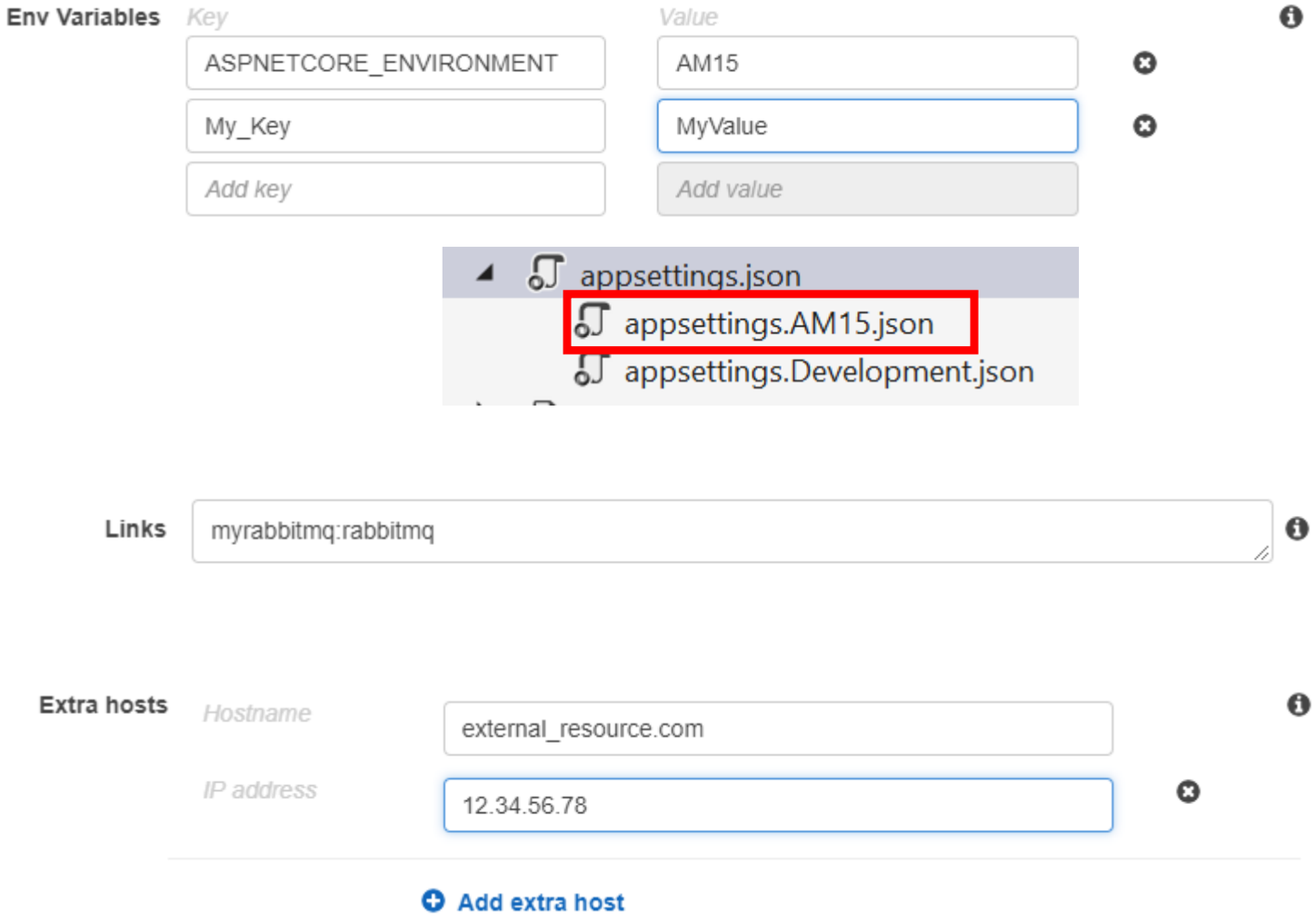


https://docs.aws.amazon.com/AmazonECS/latest/developerguide/Welcome.html

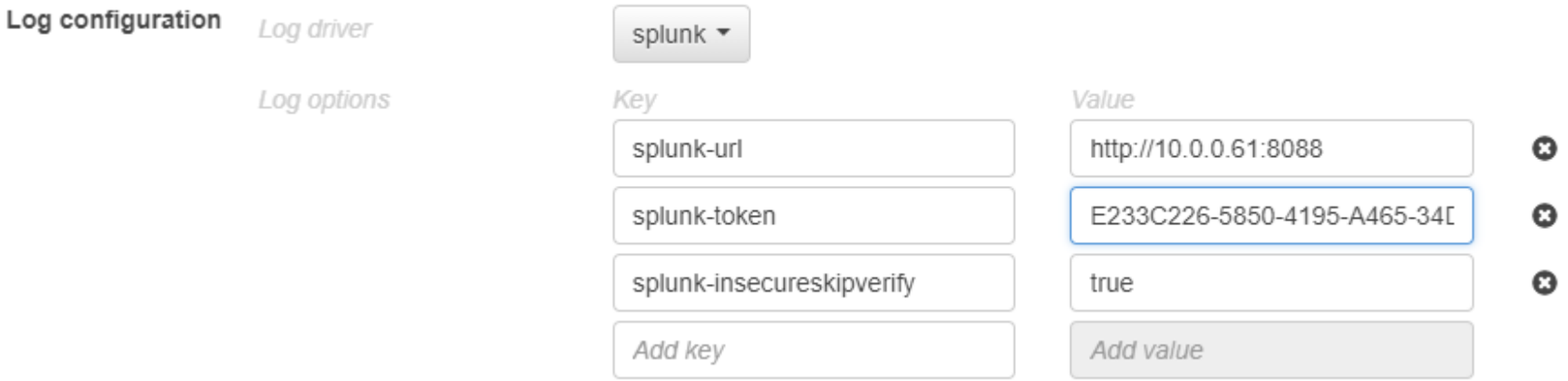
ECS. Task Definition. Параметры

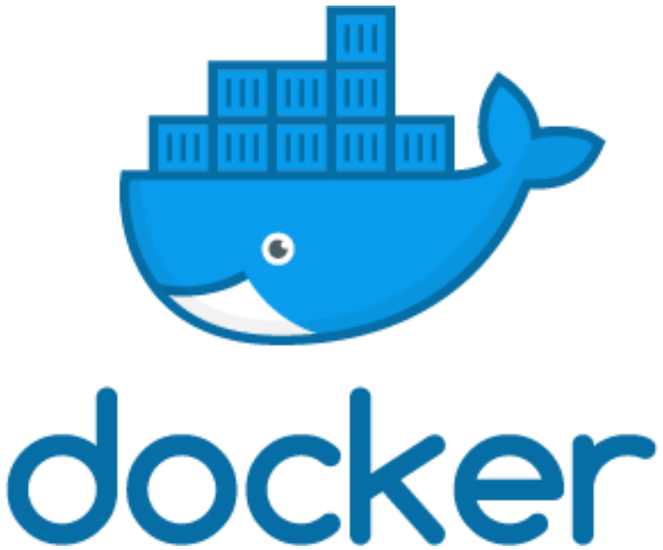


ECS. Task Definition. Параметры

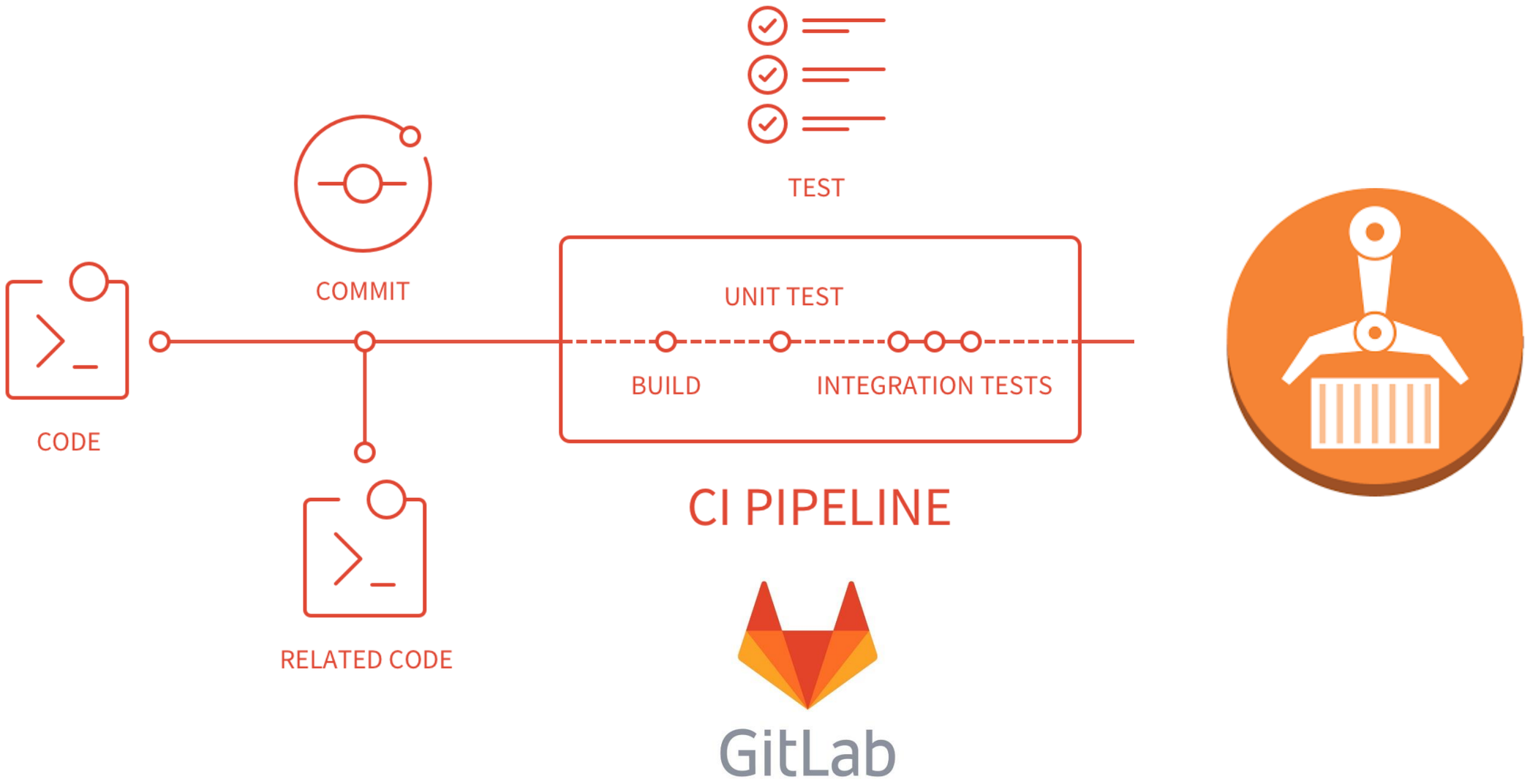


ECS. Task Definition. Логирование





GitLab CI


TEST
COMMIT
UNIT TEST
CODE
BUILD
INTEGRATION TESTS
RELATED CODE
CI PIPELINE
GitLab

GitLab runner и конфигурация

config.toml

```
concurrent = 1
check_interval = 0
[[runners]]
  name = "ASP.Net Core Runner"
  url = "<yourgitlaburl>"
  token = "<yourgitlabtoken>"
  executor = "docker"
  [runners.docker]
    tls_verify = false
    image = "microsoft/aspnetcore-build:1.0-1.1"
    privileged = true
    disable_cache = false
    volumes = ["/cache", "/var/run/docker/sock:/var/run/docker.sock"]
    shm_size = 0
    pull_policy = "if-not-present"
  [runners.cache]
```

GitLab CI .Net сборка

```
build:
    stage: build
    image: microsoft/aspnetcore-build:1.0-2.0
    script:
        - dotnet restore --configfile $CI_PROJECT_DIR/.nuget/NuGet.Config
        - dotnet publish Application -o obj/Docker/publish -c Release
        - dotnet publish Tests/Application.Tests -o obj/Docker/publish -c Release
    artifacts:
        paths:
            - Application/obj/Docker/publish
            - Tests/Application.Tests/obj/Docker/publish
        expire_in: 1h
```

GitLab CI. Запуск тестов

```
test:
    stage: test
    image: microsoft/dotnet:2.0-sdk
    script:
        - dotnet test Tests/Application.Tests --output obj/Docker/publish
            --no-build --no-restore
```

GitLab CI. Доступ к репозиторию в ECS

```
aws_login:
    stage: aws_login
    image: am15/ubuntu-aws-docker:latest
    script:
      - mkdir Application/docker
      - aws configure set aws_access_key_id ---
      - aws configure set aws_secret_access_key ---
      - aws configure set default.region eu-west-1
      - aws ecr get-login --no-include-email > Application/docker/docker-login.sh
    artifacts:
      paths:
        - Application/docker
      expire_in: 1h
    only:
        - /^(master)|(.*_deploy)$/
```

GitLab CI. Docker сборка

```
docker_push:
    stage: docker_push
    image: docker:latest
    services:
        - docker:dind
    script:
        - cd Application &&
              docker build . -t $AWS_REGISTRY_PATH/<image>:$CI_COMMIT_REF_NAME
        - chmod a+x docker/docker-login.sh
        - docker/docker-login.sh
        - docker push $AWS_REGISTRY_PATH/<image>:$CI_COMMIT_REF_NAME
    only:
        - /^(master)|(.*_deploy)$/
```

Выводы

- Работает
- Дешевле в содержании
- Переносится на другую платформу
- Дольше поначалу
- Проще, быстрее в развёртывании
- Проще для всех

Спасибо!

Александр Иванов
alexander.ivanov@arcadia.spb.ru

https://gist.github.com/ArdenIvanov/5e80188d9f179fcca8e0a2980078f1f6