Добрый день, уважаемые студенты! Я рад вас видеть на очередной лекции. Мы продолжаем предыдущую тему, а именно тему Compute. Сегодня у нас вторая часть. Итак, давайте начнем.

В четвертой секции мы с вами поговорим про сервисы, связанные с контейнерами, т.е. Container services. Далее, в пятой секции, мы поговорим про сервис AWS Lambda. И самая последняя, шестая секция, про сервис AWS Elastic Beanstalk.

Секция четвертая – Сервисы связанные с контейнерами. Перед тем как мы доберемся до контейнеров, давайте вспомним. У нас есть физический сервер. Внутри дата-центров AWS физически на определенном участке расположены максимально мощные физические сервера. Далее, благодаря виртуализации, этот физический виртуальный сервер подразделяется на независимые друг от друга виртуальные инстансы Amazon EC2. Таким образом, в рамках Amazon EC2 мы можем запрашивать более 50 различных типов, отличающихся количеством ядер процессора, а также количеством оперативной памяти.

Теперь, двигаясь еще дальше, есть еще один вариант виртуализации на уровне операционной системы, когда мы устанавливаем операционную систему и поверх операционной системы запускаем изолированный контейнер, т.е. если говорим про инстансы Amazon EC2, там вы указываете во время создания определенную операционную систему. Когда мы говорим про контейнер, контейнер привязан к определенной операционной системе и запускается в рамках этой операционной системы. Но все остальное, а именно библиотеки, все конфигурации, код, run time, среда запуска вашей программы, она вся изолирована и самодостаточна. Таким образом, мы говорим, что контейнер это что-то, что repeatable, т.е. оно легко переносимое, т.е. вы берете готовый контейнер и копируете ее, запускаете в другом месте, в другом environment-е, и она будет запускаться точно так же, так как она содержит все необходимое для своего функционирования. Еще другой важный момент, это то, что виртуальные машины запускаются намного медленнее по сравнению с контейнерами.

Теперь двигаемся дальше. Для того, чтобы создавать контейнеры, работать с ними, нам нужна определенная программа. Самая популярная программа, которая позволяет нам создавать контейнеры, это Docker, т.е. Docker-контейнер. Docker-контейнер, это такая сущность, которая содержит в себе все необходимое для запуска и корректного функционирования вашего приложения, а именно библиотеки, системные инструменты, код и среда запуска вашей программы.

Мы проговорили, что такое контейнер самыми простыми словами. Теперь давайте копнем немножко глубже. Когда мы говорим про виртуальные машины, с правой стороны вы видите пример deployment-a трех различных приложений, которые работают с различными библиотеками. Они специально выделены различным цветом, т.е. Application 1, Application 2, Application 3. Каждая из них запущена на инстансе Amazon EC2. При этом вы видите, что операционная система, она изолирована друг от друга, т.е. VM1, Virtual Machine 1, у нее своя операционная система, она может совпадать с другими, а может и нет. Это зависит от самого приложения, т.е. виртуальная машина работает поверх hypervisor-а.

Теперь следующий уровень виртуализации это контейнеры, т.е. выше hypervisor-а у нас уровень операционной системы нашего инстанса Amazon EC2. Так вот в рамках одного инстанса Amazon EC2 вы видите, с левой стороны, что мы запустили три различных контейнера. Каждый контейнер в себе содержит не только приложение, но и все, что необходимо для корректного функционирования каждого приложения. И при этом вы видите, что приложения работают с различными библиотеками, не связанные между собой. Самое главное, чтобы была нужная операционная система. Здесь необходимо также сделать пометку, что контейнеры достаточно гибкие и могут запускаться на различных операционных системах. Основное требование, чтобы были все реализованы, все необходимые фичи, функционал, который нужно для конкретного контейнера. У нас есть различные варианты операционной системы Linux. В том случае, когда для нашего контейнера нужен тот функционал, который есть у двух различных вариантов Linux операционной системы, мы говорим о том, что этот контейнер может успешно запуститься и в первом, и во втором варианте операционной системы. Таким образом, контейнер в том же состоянии, который есть, вы можете запускать на любом другом компьютере на отличающейся операционной системе. И это придает некоторую гибкость во время работы с контейнерами.

Другой момент, это то, что в рамках одного EC2 инстанса мы можем запускать очень маленькие контейнеры и в одном инстансе EC2 может находиться в один момент времени сотни различных контейнеров. Каждый контейнер обслуживает какое-то определенное приложение.

Теперь двигаясь к AWS, вы скорее всего уже в уме запланировали, что для того, чтобы работать с контейнерами, вам необходимо запустить один Amazon EC2 инстанс, там установить docker приложения и вы можете работать с контейнерами. Да, это правильный вариант работы с контейнерами в облаке, но не самый лучший. Самый лучший вариант, т.е. специально созданный для этого сервис Amazon Elastic Container Service, чаще она встречается в сокращенном варианте ECS и дает возможность вам работать с контейнерами в более удобном виде. Таким образом, вы можете запускать до 10 тысяч ваших контейнеров за несколько секунд, используя этот сервис. Вы можете также удобно мониторить, управлять и даже настраивать различные действия и расписания для запуска и управления вашими контейнерами. Здесь также следует отметить, что ECS поддерживает не только On-demand EC2 инстансы, а также Spot Instances и Reserved Instances.

Давайте рассмотрим пример. У нас есть некий task definition. Task definition это описание вашего контейнера. В нем содержится информация о вашем приложении, какие порты вы используете, дополнительные параметры возможно вы задаете для работы приложения. И представим, что в этом примере мы сделали task definition, описали два контейнера, Container A и Container B. Теперь в рамках нашей инфраструктуры нам необходимо три инстанса Container A и два инстанса Container B. Таким образом мы делаем task, т.е. задача, либо instance контейнера, если так можно выразиться. Таким образом мы вот эти маленькие инстансы, в которых сидят наши контейнеры, через task передаем сервис Amazon ECS. И этот сервис для нас в зависимости от наших дополнительных входных данных для сервиса ECS располагает эти контейнеры и запускает их в нашем кластере ECS. Cluster ECS – это набор Amazon EC2 инстансов, которые запущены в виде группы. Таким образом, сервис Amazon ECS устанавливает на каждом Amazon EC2 инстансе агент. Этот агент как раз таки помогает вам располагать ваши контейнеры внутри этих Amazon EC2 инстансов.

Когда мы говорим про ECS кластер для нас доступны три варианта. Первый вариант – это когда мы создаем наши контейнеры и описываем дополнительные параметры какой мощности должна быть этот контейнер, в каком количестве мы эти контейнеры создаем, а также необходимые параметры для настройки сети и взаимодействия этих контейнеров между собой, т.е. Networking Related Settings. В этом случае мы говорим про кейс с правой стороны и благодаря связке сервисов Amazon ECS и сервиса AWS Fargate мы можем сконцентрироваться лишь на наших контейнерах. Все остальное, т.е. управление операционной системы, управление docker-агентом, также docker-приложением, где запускаются и работают наши контейнеры, оно передается под управление AWS. Это в случае, если у вас нет ресурсов либо специалистов или нет необходимости какой-то супер тонкой настройки ваших Amazon EC2 инстансов, где запущены ваши контейнеры.

В случае же если вам нужна какая-то тонкая продвинутая настройка, она не всегда необходима. Но в этом случае вы можете воспользоваться другим вариантом слева, когда вы полностью контролируете и инстансы в рамках ECS кластера и указываете все необходимые более тонкие настройки. В этом случае есть два варианта. Ваш ECS кластер будет состоять из EC2 инстансов либо Linux, либо Windows. И в том и в этом случае, помимо того, что вы делаете в первом варианте, вам также необходимо ввести все настройки, которые необходимы для создания одного Amazon EC2 инстанса. Используя этот шаблон ваших входных данных, уже будет создаваться ECS кластер.

Давайте представим следующий кейс. Вы – компания, разрабатываете некоторое приложение и используете docker контейнеры для этого. Компания растет, через некоторое время количество контейнеров, которые у вас есть, увеличивается. И вам нужно что-то, что помогло бы вам эффективно оркестрировать, т.е. управлять вот этот большой объем ваших контейнеров. Для этого было создано open sourse software специально для container orchestration, называется Kubernetes, либо вы можете чаще встречать как K8S. Чтобы понять, что такое Kubernetes, когда мы говорим про docker, мы говорим, что мы работаем в рамках одной гостевой операционной системы. Когда мы говорим про Kubernetes, мы уже поднимаемся на уровень выше и работаем уже с несколькими Docker host-ами. Kubernetes нам позволяет упростить задачи масштабирования, container provisioning, т.е. запуска наших контейнеров, вопросы связанные с networking, а также распределением нагрузки, т.е. это то решение, которое нам помогает эффективно справляться с этим большим количеством разнородных контейнеров.

Когда мы говорим про Kubernetes, также есть некоторые популярные термины. Kubernetes управляет кластером. Cluster – это набор нескольких виртуальных машин. Каждая виртуальная машина в рамках кластера называется нодой. Мы с вами помним, что в рамках одной виртуальной машины может запускаться несколько сотен контейнеров. Так вот, в нашем случае, в случае Kubernetes, как контейнер выступают поды. Таким образом, у нас кластер состоит из нодов, и в каждой ноде может запускаться большое количество подов, т.е. контейнеров.

Мы двигаемся дальше. Теперь поговорим про следующий сервис. Вы, наверное, подумали, что вы сейчас можете поднять кластер Kubernetes следующим образом. Запускаете несколько Amazon EC2 инстансов, устанавливаете docker-приложение, поверх устанавливаете Kubernetes-приложение, и вот у вас есть Kubernetes-cluster, с которым вы можете работать. Да, это действительно так, это один из вариантов, но когда мы работаем с AWS, у нас есть вариант получше, а именно сервис Amazon Elastic Kubernetes Service, чаще вы его будете встречать как Amazon ЕKS. Это тот сервис, который предоставляет нам Managed Kubernetes Service, т.е. это тот сервис, который полностью совместим с приложением Kubernetes, и сервис позволяет пользователям, клиентам нашего облачного провайдера AWS разгрузиться, и большую часть операционной деятельности по обслуживанию этого кластера передать AWS, а самим больше сконцентрироваться на бизнес-задачах. То, что этот сервис совместим с приложением Kubernetes, говорит о том, что мы наши нагрузки на Kubernetes, уже запущенные на локальном дата-центре, можем с легкостью перенести на AWS, а именно в сервис Amazon Elastic Kubernetes Service.

У вас может возникнуть вопрос «Чем же все-таки отличается сервис Amazon ECS, а также Amazon ЕKS?». На самом деле оба варианта, оба сервиса помогают нам оркестрировать нашим кластером Docker-контейнеров. Отличие лишь в том, что в первом случае мы работаем непосредственно с Docker-контейнерами и управляем нашим кластером посредством сервиса AWS. Когда мы говорим про ЕKS, оркестрацием кластера уже занимается не решение от AWS, а open sourсe программа Kubernetes, но она обернута в сервис, который нас разгружает от некоторых операционных моментов. Они похожи в зависимости от того, что у вас установлено, как ваша инфраструктура поднята, как она функционирует, либо если сейчас вы ни то, ни другое не используете, то как минимум у клиентов AWS есть возможность выбрать один из этих вариантов.

Следующий сервис, о котором я хотел вам рассказать, это Amazon Elastic Container Registry, чаще вы его будете встречать как Amazon ECR. Это тот сервис, который является хранилищем всех ваших Docker-образов. Таким образом, когда вы создаете ваш кластер, неважно с использованием сервиса Amazon ECS, либо с использованием сервиса Amazon ЕKS, вам необходимо указать ваш контейнер, образ этого контейнера. В этом случае необходимо воспользоваться сервисом Amazon ECR. Как аналогия могу привести, когда мы с вами создаем Amazon EC2 инстанс, мы указываем AMI, т.е. образ вашего будущего инстанса. AMI в этом случае выступает как Docker-образ. И хранилище всех ваших AMI в случае с контейнерами – это сервис Amazon ECR.

Мы с вами добрались до конца четвертой секции. Давайте пройдемся по самым основным моментам. Контейнеры – это нечто, что может в себе хранить все необходимое для успешного запуска вашего приложения. Сюда входят библиотеки, системные настройки, код и так далее. Docker – это та программа, которая позволяет вам создавать контейнеры. Это одна из самых популярных программ для этого. Одно приложение может запускаться в нескольких контейнерах, связанных между собой. Есть сервис Amazon Elastic Container Service, т.е. ECS, который позволяет вам оркестрировать вашими Docker-контейнерами. Следующее популярное приложение, программа – это Kubernetes. Это open source решение, которое позволяет вам оркестрировать ваши контейнеры. Специально для Kubernetes был создан отдельный сервис Amazon Elastic Kubernetes Service. Он совместим с Kubernetes и позволяет вам разгрузиться от операционной работы, но при этом управлять вашим кластером Docker-контейнеров через Kubernetes. И третий сервис это Amazon Elastic Container Registry. Это сервис, который является хранилищем всех ваших Docker-контейнеров.

Мы с вами добрались до пятой секции и здесь мы поговорим про сервис AWS Lambda. Это мой самый любимый сервис внутри всех сервисов AWS и сейчас вы узнаете почему. Мы с вами ранее проговаривали, что у нас есть различные сервисы, которые предоставляют различные IT-ресурсы. Когда мы говорим про Compute, Amazon EC2 сервис предоставляет нам виртуальные машины. Сервисы Amazon ECS и Amazon EKS помогают нам работать с контейнерами. Так вот, следующий уровень – это AWS Lambda, Serverless Computing, т.е. это сервис, который исключает абсолютно все операционные задачи под вас и достаточно вам ваш код загрузить в этот сервис. Все что связано с запуском этого сервиса, поддержкой, настройкой мониторинга, и так далее, занимается и берет на себя AWS. Таким образом, я как разработчик могу исключить необходимость системного администратора при построении какого-то решения. Также и вы, так как IT университеты больше выпускают разработчиков, нежели системных администраторов, вы также можете воспользоваться этим сервисом для запуска своих собственных решений, собственных стартап-проектов, где благодаря AWS исключается необходимость целого человека, специалиста, т.е. системного администратора. Это поможет вам сократить некоторые расходы и увеличить шансы успешного запуска вашего стартапа.

Касательно оплаты, здесь тоже очень важный момент. Вы оплачиваете за количество времени, когда AWS Lambda была запущена. В случае, когда ваш код не запускается, вы абсолютно ничего не оплачиваете. Это также идеально подходящая модель оплаты для тех же стартапов, либо для тех нагрузок, которые не постоянны. Представьте, вы некий стартап, у вас нагрузки небольшие, потому что вы не наработали клиентскую базу, вы не настолько популярны, таким образом ваш код будет запускаться только в тот момент, когда ваше приложение будут использовать. Если ваше приложение никто не использует, то вы соответственно ничего не оплачиваете.

Давайте рассмотрим подробнее, какие есть преимущества у сервиса AWS Lambda. С AWS Lambda вам нет необходимости изучать новые языки программирования. Оно и так изначально поддерживает самые популярные языки программирования, в том числе Python, JavaScript, Node.js, Ruby, C# и т.д. Вы уже поняли, что большую часть администрирования по запуску вашей Lambda-ы берет на себя AWS, вам необходимо лишь загрузить ваш код. Внутри AWS также встроена защита от падений. Что это значит? Это значит, микроконтейнеры, в котором запускается ваш код, находятся в нескольких availability zone-ах в рамках региона. Таким образом, если что-то происходит в availability zone-e, то ваш контейнер запустится в другой availability zone-e . Вы этого даже никак не заметите и не почувствуете, не узнаете, оно встроенно сидит внутри AWS Lambda. Таким образом, все ваши запросы на AWS Lambda будут успешно отработаны. Бывают некоторые нагрузки, когда вам необходимо запускать несколько AWS Lambda, это какая-то сложная логика и одна лямбда может запускать другую лямбду, либо в зависимости от результатов выполнения одной лямбды, при некоторых дополнительных условиях вы можете запускать или не запускать другие лямбды. Для построения этой сложной логики, либо оркестрации нескольких лямбда-функций, вы можете воспользоваться сервисом AWS Step Functions. AWS Step Functions как раз таки расписывает эту логику и в зависимости от логики она будет вызывать ту или иную лямбду. Но лямбда также может использоваться отдельно, самостоятельно, независимая функция, которая в зависимости от запроса возвращает какой-то готовый ответ. Например, у вас есть веб-сайт и вы запрашиваете информацию о каком-то продукте. Таким образом, когда запрос прилетает до вашей лямбды, лямбда, используя входные данные, идет в хранилище информации о ваших продуктах и извлекает необходимую информацию, далее возвращает это вызывающей стороне. Как хранилище у вас может выступить объектное хранилище в качестве сервиса AWS S3. Если у вас SQL база данных, это сервис AWS RDS, в случае если у вас NoSQL база данных, то у нас есть AWS DynamoDB, т.е. вариантов интеграции сервиса AWS Lambda с другими сервисами имеется большое количество. Также немаловажное преимущество это pay-per-use pricing, т.е. вы оплачиваете только за то время, когда ваша лямбда была запущена. Это позволяет значительно сэкономить ваши затраты, в случае если ваши нагрузки не постоянны.

Давайте поговорим о самых популярных сервисах, которые работают в связке с AWS Lambda. Когда мы говорим про объектное хранилище, самым популярным вариантом является Amazon S3. Он нативно интегрируется с этим сервисом и никаких проблем не возникает. Другой вариант, когда мы говорим, что нам необходимо хранить некоторые данные в не структурированном виде, нам необходимо использовать NoSQL баз данных. В этом случае выступает Amazon DynamoDB. Amazon DynamoDB является также serverless решением баз данных NoSQL. Таким образом, она также нативно интегрирована с сервисом AWS Lambda.

Далее у нас есть два сервиса Amazon SNS, а также Amazon SQS. Amazon SNS – это когда мы в зависимости от тех или иных условий можем в формате Push отправлять уведомления. Когда мы говорим Amazon SQS, это работа с очередью и нагрузки формата Pull, т.е. в случае Amazon SNS мы передаем какое-то сообщение, а оно сразу отправляется, push-ится к получателям. Когда мы говорим Amazon SQS, это pull, т.е. в Amazon SQS в очереди накапливается определенное количество сообщений и мы в режиме pull, т.е. подтягиваем необходимый объем сообщений для обработки в тот момент, когда нам это нужно. Это два варианта построения архитектуры в облаке и оба варианта также интегрированы с AWS Lambda.

Когда мы говорим, что нам необходимо построить API, т.е. Application Program Interface, мы можем интегрироватьAWS Lambda с Amazon API Gateway. Это одна из самых популярных связок AWS Lambda.

Еще один существующий, но менее популярный вариант сервиса – это Application Load Balancer. Например, представьте, у вас есть кластер EC2, он запускается, делает некоторые нагрузки, но при этом вы знаете, что после восьми часов вечера вам смело можно отключать 50% ваших EC2 инстанcов, потому что нагрузка резко уменьшается. В этом случае вы можете настроить event, которое триггерит вашу лямбду в 8 часов вечера, дальше AWS Lambda уже передает запрос на сервис Application Load Balancer, который сокращает количество инстанcов Amazon EC2.

Давайте теперь подробнее поговорим о том, какие настройки необходимо произвести для запуска нашей лямбда-функции. Самое первое, нам нужен код программы, который будет запускаться. Далее нам необходимо настроить IAM role для лямбда-функции. Если вы создаете лямбда-функцию в AWS Management Console, то для вас создается role с минимальными правами, а именно с правами для сервиса AWS CloudWatch, чтобы функция могла писать туда свои логи отработки. Если вам необходимо, чтобы лямбда-функция работала с другими AWS сервисами, вам необходимо соответствующие policy, права, доступы добавить в эту role. Также нам необходимо указать Run time, т.е. в какой среде запускать ваш код, либо это Python, Node.js, C# и т.д.

Помимо самого кода AWS Lambda, мы также можем использовать другие зависимости. Это могут быть дополнительные кастомные библиотеки, необходимые для запуска вашего кода. В этом случае вам необходимо создать архив и этот архив загружать в сервисы AWS Lambda. В случае, когда вы не используете дополнительные библиотеки, библиотеки достаточно редко используются и в случае Python-а, стандартных библиотек Python-а хватает для большинства задач. В этом случае у нас есть только код программы и вы можете работать в Console AWS Lambda-ы и копировать, вставлять код и сохранять этот код вот в этой же странице, что очень удобно.

Касательно времени запуска, функция максимум может запускаться 15 минут, т.е. 900 секунд. Больше этого она запускаться не может. В случае, если вам нужно больше времени, вам необходимо рассмотреть вариант использования Step Functions, когда одна лямбда запускает следующую лямбду, таким образом вы получаете дополнительные 15 минут запуска. В случае, если этот вариант вам не подходит, вы можете также посмотреть в сторону контейнеров, либо в сторону инстанcов EC2. В зависимости от вашей бизнес-задачи тот или иной вариант может быть вам ближе либо выгоден с точки зрения реализации.

Когда мы говорим о мощности нашей функции, то мы контролируем количество оперативной памяти, которую мы можем ей выделить. Мы можем выдать минимально 128 МБ для этой функции, либо до 10 ГБ оперативной памяти. Количество ядер будет увеличиваться пропорционально увеличению количества оперативной памяти. Ее мы отдельно от оперативной памяти устанавливать не можем.

Давайте рассмотрим пример использования AWS Lambda, достаточно популярный пример. У нас есть набор EC2 инстанcов и в зависимости от наших нагрузок мы понимаем, что в 10 часов вечера мы можем отключать все наши инстансы, а в 5 часов утра нам необходимо стартовать наши инстансы. Как мы можем настроить нашу архитектуру для того, чтобы это все работало. Мы создаем два event-a. Эти event-ы создаются в сервисе AWS CloudWatch. Первый event по расписанию в 10 часов вечера будет триггерить первую лямбду. Эта лямбда на вход принимает этот event. Также в этой лямбде либо в event-е мы можем прописать список инстанcов на отключение и в момент запуска этой у нее есть соответствующая rоle, у этой rоle есть права для отключения из этого инстансов. Она запускает соответствующие команды и наши инстансы отключаются.

Второй event по расписанию триггерит в 5 часов утра уже другую вторую лямбду. У этой лямбды есть соответствующая либо та же rоle, либо другая rоle, у которой есть права на запуск истинсов. Так вот в момент триггера event-ом AWS CloudWatch нашей Lambda функции, она исполняет свой код и в зависимости от входных данных списка инстанcов, эти инстанcы она стартует.

Давайте теперь рассмотрим второй пример. Представим, что у нас есть некоторое приложение, которое в себе содержит фотографии. Наш пользователь приложения загружает некоторую фотографию. Эта фотография у нас сохраняется в Amazon S3 bucket. Далее как только у нас файл загрузился в Amazon S3 bucket, запускается event, который триггерит нашу лямбду. Запускается наша лямбда, в event-е сидят входные данные, а в том числе информация с ссылкой на эту картинку. Мы понимаем какая картинка была загружена. Далее Lambda использует свою rоle, у нее есть соответствующие права. Она загружает эту картинку, обрабатывает ее и создает эту же картинку в различных разрешениях. В том числе и картинку так называемую thumbnail. Это мини версия этой картинки, которая весит очень мало, но при этом эта картинка отображается, когда мы видим наши картинки в списке. Далее после отработки, подготовки всех необходимых картинок, она загружает уже эти картинки в другой Amazon S3 bucket. И если додумывать дальше, то приложение уже может отображать другим пользователям список этих фотографий в различных разрешениях, а также может использовать thumbnail для отображения мини версии при просмотре списка всех наших картинок.

Давайте подробнее остановимся на основных лимитах сервиса AWS Lambda. Здесь хотел бы также отметить, большинство лимитов, она является soft и может быть увеличена по обращению в AWS Support. Вы можете в рамках одного региона запускать параллельно до 1000 execution-ы Lambda. Представим, что у вас есть определенная функция, она передает информацию о каком-то продукте и в случае, если внезапно 1000 пользователей запросят информацию о некотором продукте и вам необходимо параллельно запустить 1000 лямбд, то AWS на своей стороне автоматически создаст микроконтейнеры, где запускаются ваши лямбды в необходимом количестве до 1000 и успешно запустит все эти лямбды. Далее, если количество запросов резко уменьшится, то все запущенные микроконтейнеры будут постепенно уничтожаться по внутренней логике AWS. В случае, если у вас придет больше запросов, то вы можете настроить retry логику и ваш запрос будет через некоторое время отправлен еще раз. Так как на тот момент уже будет скорее всего запускаться меньшее количество Lambda, то эти запросы тоже будут успешно обработаны.

Если мы говорим про размер нашей функции, включая код, а также все необходимые библиотеки для запуска, то оно может быть размером до 250 мегабайтов в не архивированном состоянии. Если мы говорим про мощность Lambda, то мы можем выделить минимум 128 мегабайтов оперативной памяти и максимум 10 гигабайтов оперативной памяти для одной нашей функции. Здесь следует отметить, что количество ядер процессора будет выделяться параллельно количеству выделенных мегабайтов оперативной памяти. Если мы говорим про максимальную длительность запуска одной функции, то это 900 секунд либо 15 минут.

Мы с вами добрались до конца пятой секции. Давайте пройдемся по самым основным моментам. AWS Lambda – это тот сервис, который предоставляет нам бессерверные мощности. Идея в том, что мы не менеджим наши серверы, а лишь загружаем наш код. Все, что относится к обслуживанию запуска этого кода, оно происходит на стороне AWS. Также AWS Lambda включает автоматическое масштабирование, т.е. если внезапно происходит всплеск запросов на нашу лямбду, то все эти запросы параллельно будут обработаны. AWS Lambda может быть достаточно мощной и мы можем выделять до 10 гигабайтов оперативной памяти для одной нашей функции. Более того, каждая наша Lambda может запускаться максимум 15 минут. Этого времени более чем достаточно для выполнения большинства задач. На этом мы остановились на самых основных моментах, связанных с сервисом AWS Lambda. Я очень надеюсь, что этот сервис вам пригодится в будущем при построении ваших стартапов. Это идеальный выбор для запуска чего-либо в облаке. И также никогда не забывайте, что нельзя сильно привязываться к Lambda. И бывают ситуации, когда нужно ваши нагрузки, которые стали постоянными, переносить уже на другие мощности. Но когда мы говорим про самое начало, когда ничего не понятно, ничего не известно, то все serverless сервисы, начиная от лямбды, заканчивая другими сервисами, работающими в связке с этой лямбдой, являются идеальным решением.

На этом мы с вами добрались до шестой секции. Последняя секция в рамках нашей сегодняшней лекции. И мы с вами поговорим подробнее про сервис AWS Elastic Beanstalk. AWS Elastic Beanstalk является следующим примером Compute сервиса, предоставляется в форме PaaS, т.е. Platform as a Service. Идея в том, что вы загружаете код вашего веб-приложения, AWS Elastic Beanstalk поднимает всю необходимую инфраструктуру для того, чтобы запустить это веб-приложение. Что подразумевается под поднять инфраструктуру, это фактически развертывание вашего веб-приложения, настройка балансирования нагрузки, а также автоматического масштабирования и все необходимое связанное с мониторингом и логированием вашего веб-приложения. Здесь следует отметить, что использование этого сервиса является бесплатным, т.е. вы ни доллара не оплачиваете за этот сервис. Но все те ресурсы, которые были подняты в рамках сервиса AWS Elastic Beanstalk, они оплачиваются по стандартным тарифам. Например, если вы передали некоторые входные параметры в сервис AWS Elastic Beanstalk и для вас была поднята инфраструктура с двумя инстансами Amazon EC2, то вы будете оплачивать за эти два инстанса Amazon EC2, по стандартному тарифу этого сервиса. Это и относится ко всем другим IT-ресурсам и другим сервисам AWS, с которыми взаимодействует AWS Elastic Beanstalk, т.е. вы бы столько же денег заплатили бы, если бы вы поднимали те же два инстанса Amazon EC2 вручную. Разница лишь в том, что поднятие инфраструктуры было проделано вместо вас, для вас, автоматически через сервис AWS Elastic Beanstalk. Фактически вы оплачиваете ту же сумму, плюс ко всему этому экономите свое время и поднимаете инфраструктуру автоматически.

Здесь вы можете наглядно видеть, какую часть работы AWS через сервис Elastic Beanstalk берет на себя, т.е. то, что вы управляете, это ваш код и ваши настройки для AWS Elastic Beanstalk. Все остальное управляется AWS, т.е. HTTP-сервер, application server, language interpreter, т.е. среда запуска вашего кода, операционная система и даже host. Вы можете использовать AWS Elastic Beanstalk как в AWS Management Console, также вы можете использовать AWS CLI для запуска инфраструктуры с использованием AWS Elastic Beanstalk. Когда мы говорим про поддержку платформ, то поддерживаются Docker, Go, Java, .NET, Node.js, PHP, Python и Ruby и другие платформы. Если мы говорим про веб-серверы, которые поддерживаются, то для Java приложений поддерживается Apache Tomcat, для PHP и Python приложения поддерживается Apache HTTP Server. Если мы говорим про Node.js приложения, для них доступны NGINX и Apache HTTP Server. Если Ruby приложения, то это Passenger и Puma. Если мы говорим про .NET приложения, Java и Docker вместе с Go, то поддерживается Microsoft Internet Information Services, т.е. IIS.

Давайте остановимся на основных преимуществах сервиса AWS Elastic Beanstalk. Первое это то, что вы очень быстро и просто можете начать, т.е. запустить ваше веб-приложение, поднимается вся необходимая инфраструктура, вы фактически меньше времени теряете на все это дело. Другой момент это то, что ваши специалисты, ваши разработчики освобождаются от операционной деятельности и могут сконцентрироваться на бизнес-задачах. Таким образом, эффективность ваших разработчиков повышается. Другой момент это то, что AWS Elastic Beanstalk подходит для большинства веб-приложений. Таким образом, очень редко, когда возможности AWS Elastic Beanstalk для веб-приложения становятся мало. И в большинстве случаев вы будете двигаться с AWS Elastic Beanstalk и все будет хватать. Другой момент это то, что в AWS Elastic Beanstalk вы можете достаточно гибко настроить ваши ресурсы. Как пример, вы можете указать определенный instance type в рамках сервиса Amazon EC2, который может поднимать Elastic Beanstalk для вас.

На этом мы подошли к концу шестой секции. Давайте остановимся на основных моментах. В случае, когда у вас веб-приложение, то обязательно следует рассмотреть сервис AWS Elastic Beanstalk, так как она помогает вам упростить процесс развертывания вашей IT-инфраструктуры в облаке. AWS Elastic Beanstalk поддерживает достаточно большой выбор платформ — это Java,.NET, PHP, Node.js, Python, Ruby, Go, Docker и другие. Если мы говорим касательно оплаты, это сервис полностью бесплатный для клиентов AWS и вы не оплачиваете за использование этого сервиса. Но вам необходимо оплачивать по стандартным тарифам за те сервисы, где вы создаете ее ресурсы. Например, если AWS Elastic Beanstalk поднимает инстансы Amazon EC2, то да, вы за эти инстансы Amazon EC2 в рамках стандартных тарифов будете оплачивать за использование.

На этом мы подошли к концу нашей сессии. Давайте остановимся на самых основных моментах. За эти две лекции мы с вами рассмотрели, какие сервисы вычисления для нас доступны в AWS. Мы подробнее познакомились с сервисом Amazon EC2. Далее мы посмотрели, какие сервисы есть, когда мы работаем с контейнерами. Это Amazon EKS, Amazon ECS и Amazon ECR. Далее познакомились с serverless решением. Это AWS Lambda. Рассмотрели, какие плюсы есть, минусы. Ну и в самом конце познакомились с сервисом для быстрого развертывания веб-приложении. Это AWS Elastic Beanstalk. Вы видите, что AWS предоставляет широкий выбор сервисов и в зависимости от вашей бизнес потребности для вас могут подойти один или несколько вариантов, которые вы будете использовать. Оно будет максимально эффективно и максимально выгодно для вас.

Если вам необходима дополнительная информация по тому или иному сервису, здесь вы видите дополнительные ссылки, которые вам могут быть полезны.

На этом мы завершаем наше сегодняшнее лекционное занятие. Я надеюсь, вы научились чему-то новому. Увидимся с вами на следующих наших активностях.