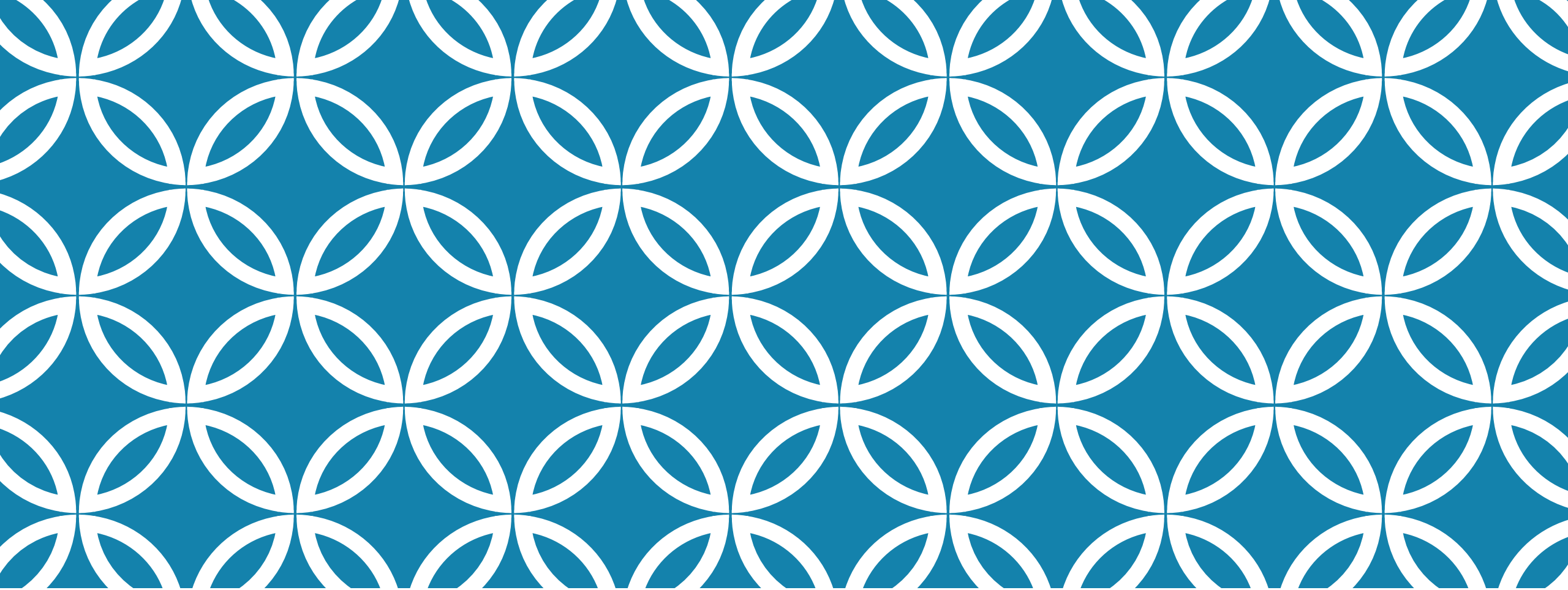


МОДУЛЬ 14. АВТОМАТИЗАЦИЯ СЕТИ

КАФЕДРА ТЕЛЕКОММУНИКАЦИЙ

14.1 ОБЗОР АВТОМАТИЗАЦИИ
14.1.1 УВЕЛИЧЕНИЕ АВТОМАТИЗАЦИИ

Вот некоторые из преимуществ автоматизации:

В результате автоматизации производительность существенно повышается, так как машины могут работать 24 часа в сутки без перерывов.

Машины также повышают единообразность выпускаемой продукции.

Автоматизация позволяет собирать огромные объемы данных, которые можно быстро проанализировать и получить информацию, полезную для управления событием или процессом.

Роботы используются в опасных условиях, например в шахтах, при тушении пожаров и устранении промышленных аварий. Это снижает риск для людей.

При определенных обстоятельствах интеллектуальные устройства могут изменить свое поведение, чтобы уменьшить потребление энергии, поставить медицинский диагноз и повысить безопасность вождения автомобиля.

14.1.2 УМНЫЕ УСТРОЙСТВА

Многие устройства теперь включают интеллектуальные технологии, помогающие управлять их поведением. Это может быть простое снижение потребляемой мощности смарт-устройством в периоды пиковой нагрузки или что-то сложное, например самостоятельное движение автомобиля.

Если устройство принимает решение или выполняет действие на основе внешней информации, это устройство считается интеллектуальным. В названиях многих устройств, с которыми мы взаимодействуем в настоящее время, есть слово «смарт», то есть «умный», «интеллектуальный». Оно указывает, что устройство может изменять свое поведение в зависимости от окружающей среды.

Чтобы устройства могли «мыслить», их необходимо программировать с использованием средств автоматизации сети.

14.2 УМНЫЕ УСТРОЙСТВА
14.2.1 КОНЦЕПЦИЯ ФОРМАТОВ ДАННЫХ

Форматы данных - это просто способ хранения и обмена данными в структурированном формате. Один из таких форматов Hypertext Markup Language (HTML). HTML является стандартным языком разметки для описания структуры веб-страниц.

Вот некоторые распространенные форматы данных, которые используются во многих приложениях, включая автоматизацию сети и программируемость:

JavaScript Object Notation (JSON)

eXtensible Markup Language (XML)

YAML Yet Anoter Markup Language (YAML)

Выбранный формат данных будет зависеть от формата, используемого приложением, инструментом или скриптом, которые вы используете. Многие системы смогут поддерживать более одного формата данных, что позволяет пользователю выбирать формат.

14.2.2 ПРАВИЛА ФОРМАТОВ ДАННЫХ

Форматы данных имеют правила и структуру, аналогичные тем, которые мы имеем в программировании и письменных языках. Каждый формат данных будет иметь определенные характеристики:

- Синтаксис, который включает типы используемых скобок, такие как [], (), {}, использование пробелов или отступов, кавычек, запятых и т. д.

- Как должны быть представлены объекты, такие как символы, строки, списки и массивы.

Как должны быть представлены пары ключ / значение. Ключ обычно находится слева, и он идентифицирует или описывает данные. Значение справа представляет собой сами данные и может быть символом, строкой, числом, списком или данными другого типа.


```
{"message": "success", "timestamp": 1560789216, "iss_position": {"latitude": "25.9990",
"longitude": "-132.6992"}}
```

14.2.3 СРАВНЕНИЕ ФОРМАТОВ ДАННЫХ

```
{
 "message": "success",
   "timestamp": 1560789260,

   "iss_position": {
 "latitude": "25.9990",
"longitude": "-132.6992"
   }
}
```

Формат JSON

```
message: success
timestamp: 1560789260
iss_position:
   latitude: '25.9990'
longitude: '-132.6992'
```

Формат YAML

```
<?xml version="1.0" encoding="UTF-
8" ?> <root>
<message>success</message>
<timestamp>1560789260</timestamp>
<iss_position>
   <latitude>25.9990</latitude>
   <longitude>-
132.6992</longitude>
</iss_position>
</root>
```

Формат XML

14.2.4 ФОРМАТ ДАННЫХ JSON

JSON - это читаемый человеком формат данных, используемый приложениями для хранения, передачи и чтения данных. JSON - очень популярный формат, используемый веб-сервисами и API для предоставления общедоступных данных. Это потому, что его легко анализировать, и его можно использовать с большинством современных языков программирования, включая Python.

14.2.4 ФОРМАТ ДАННЫХ JSON

```
GigabitEthernet0/0/0 is up, line
protocol is up (connected)
   Description: Wide Area Network
   Internet address is 172.16.0.2/24
```

Сравните вывод IOS выше с выводом в формате JSON справа. Обратите внимание, что каждый объект (каждая пара ключ/значение) представляет собой отдельный фрагмент данных об интерфейсе, включая его имя, описание и то, включен ли интерфейс.

```
{
"ietf-interfaces:interface": {
 "name": "GigabitEthernet0/0/0",
    "description": "Wide Area
Network”,
    "enabled": true,
    "ietf-ip:ipv4": {
        "address": [
            {
            "ip": "172.16.0.2",
            "netmask": "255.255.255.0"
            }
        ]
    }
  }
}
```

14.2.5 ПРАВИЛА СИНТАКСИСА JSON

Вот некоторые из характеристик JSON:

- Он использует иерархическую структуру и содержит вложенные значения.

- Он использует фигурные скобки { } для хранения объектов и квадратные скобки [ ] для хранения массивов.

- Его данные записываются в виде пар ключ/значение.

В JSON данные, известные как объект, представляют собой одну или несколько пар ключ/значение, заключенных в фигурные скобки { }. Синтаксис для объекта JSON включает в себя:

- Ключи должны быть строками в двойных кавычках " ".

- Значения должны быть допустимым типом данных JSON (строка, число, массив, логический, нулевой или другой объект).

- Ключи и значения разделяются двоеточием.

- Несколько пар ключ/значение внутри объекта разделяются запятыми.

- Пробел не имеет значения.

14.2.5 ПРАВИЛА СИНТАКСИСА JSON

Иногда ключ может содержать более одного значения. Это называется массивом. Массив в JSON представляет собой упорядоченный список значений. Характеристики массивов в JSON включают в себя:

- За ключом следует двоеточие и список значений в квадратных скобках [ ].

- Массив представляет собой упорядоченный список значений.

- Массив может содержать несколько типов значений, включая строку, число, логическое значение, объект или другой массив внутри массива.

- Каждое значение в массиве отделяется запятой.

14.2.5 ПРАВИЛА СИНТАКСИСА JSON

Например, список адресов IPv4 может выглядеть следующим образом. Ключ - “addresses”.

Каждый элемент в списке является отдельным объектом, разделенным фигурными скобками { }.

Объектами являются две пары ключ/значение: адрес IPv4 (“ip”) и маска подсети (“netmask”), разделенные запятой.

Массив объектов в списке также разделяется запятой после закрывающей скобки для каждого объекта.

```
{
   "addresses": [
       {
 "ip": "172.16.0.2",
 "netmask": "255.255.255.0"
       },
       {
 "ip": "172.16.0.3",
 "netmask": "255.255.255.0"
       },
       {
 "ip": "172.16.0.4",
 "netmask": "255.255.255.0"
       }
   ]
}
```

14.2.6 ФОРМАТ ДАННЫХ YAML

YAML - это другой тип читаемого человеком формата данных, используемый приложениями для хранения, передачи и чтения данных. Некоторые характеристики YAML:

- Похож на JSON и считается надмножеством JSON.

- YAML имеет минималистский формат, облегчающий чтение и запись.

- Он использует отступ для определения своей структуры, без использования скобок или запятых.

14.2.6 ФОРМАТ ДАННЫХ YAML

```
ietf-interfaces:interface:
 name: GigabitEthernet2
 description: Wide Area Network
 enabled: true
   ietf-ip:ipv4:
      address:
          - ip: 172.16.0.2
       netmask: 255.255.255.0
          - ip: 172.16.0.3
       netmask: 255.255.255.0
          - ip: 172.16.0.4
       netmask: 255.255.255.0
```

Вывод IOS в формате JSON находится справа. Те же данные в формате YAML приведены слева. Его легче читать.

Подобно JSON, объект YAML представляет собой одну или несколько пар ключ/значение. Пары ключ/значение отделяются двоеточием без использования кавычек. В YAML дефис используется для разделения каждого элемента в списке.

```
{
   "ietf-
interfaces:interface": {
 "name": "GigabitEthernet2",
 "description": "Wide Area
Network",
 "enabled": true,
      "ietf-ip:ipv4": {
         "address": [
            {
 "ip": "172.16.0.2",
 "netmask": "255.255.255.0"
            },
            {
 "ip": "172.16.0.3",
 "netmask": "255.255.255.0"
            },
            {
 "ip": "172.16.0.4",
 "netmask": "255.255.255.0"
            }
         ]
      }
   }
}
```

14.2.7 ФОРМАТ ДАННЫХ XML

XML - это еще один тип читаемого человеком формата данных, который используется для хранения, передачи и чтения данных приложениями. Некоторые из характеристик XML:

- Похож на HTML, который является стандартизированным языком разметки для создания веб-страниц и веб-приложений.

- XML самоописательный. Он заключает данные в связанный набор тегов.**<tag>data</tag>**

- В отличие от HTML, XML не использует предопределенные теги или структуру документа.

Объекты XML - это одна или несколько пар ключ/значение, в качестве имени ключа используется начальный тег: **<key>value</key>**

14.2.7 ФОРМАТ ДАННЫХ XML

Следующий вывод показывает те же данные для GigabitEthernet2, отформатированные как структура данных XML.

Обратите внимание, как значения заключены в теги объекта. В этом примере каждая пара ключ/значение находится на отдельной строке, а некоторые строки имеют отступ. Это не обязательно, но сделано для удобства чтения.

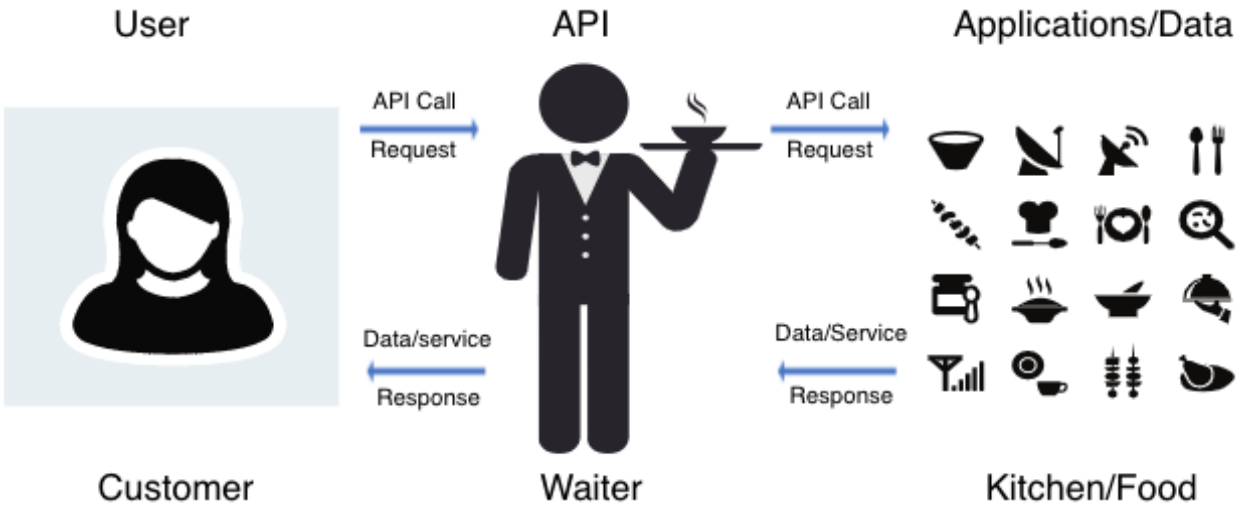
Список использует повторяющиеся экземпляры **<tag></tag>** для каждого элемента в списке. Элементы в этих повторяющихся экземплярах представляют одну или несколько пар ключ/значение.

```
<?xml version="1.0" encoding="UTF-8" ?>
<ietf-interfaces:interface>
   <name>GigabitEthernet2</name>
   <description>Wide Area
Network</description>
   <enabled>true</enabled>
   <ietf-ip:ipv4>
      <address>
         <ip>172.16.0.2</ip>
         <netmask>255.255.255.0</netmask>
      </address>
      <address>
         <ip>172.16.0.3</ip>
 <netmask>255.255.255.0</netmask>
      </address>
      <address>
         <ip>172.16.0.4</ip>
         <netmask>255.255.255.0</netmask>
      </address>
   </ietf-ip:ipv4>
</ietf-interfaces:interface>
```

14.3 API
14.3.1 ОСНОВЫ API

API - это программное обеспечение, которое позволяет другим приложениям получать доступ к его данным или услугам. Это набор правил, описывающих, как одно приложение может взаимодействовать с другим, и инструкции, позволяющие этому взаимодействию происходить. Пользователь отправляет запрос API на сервер, запрашивая конкретную информацию, и получает в ответ API от сервера вместе с запрошенной информацией.

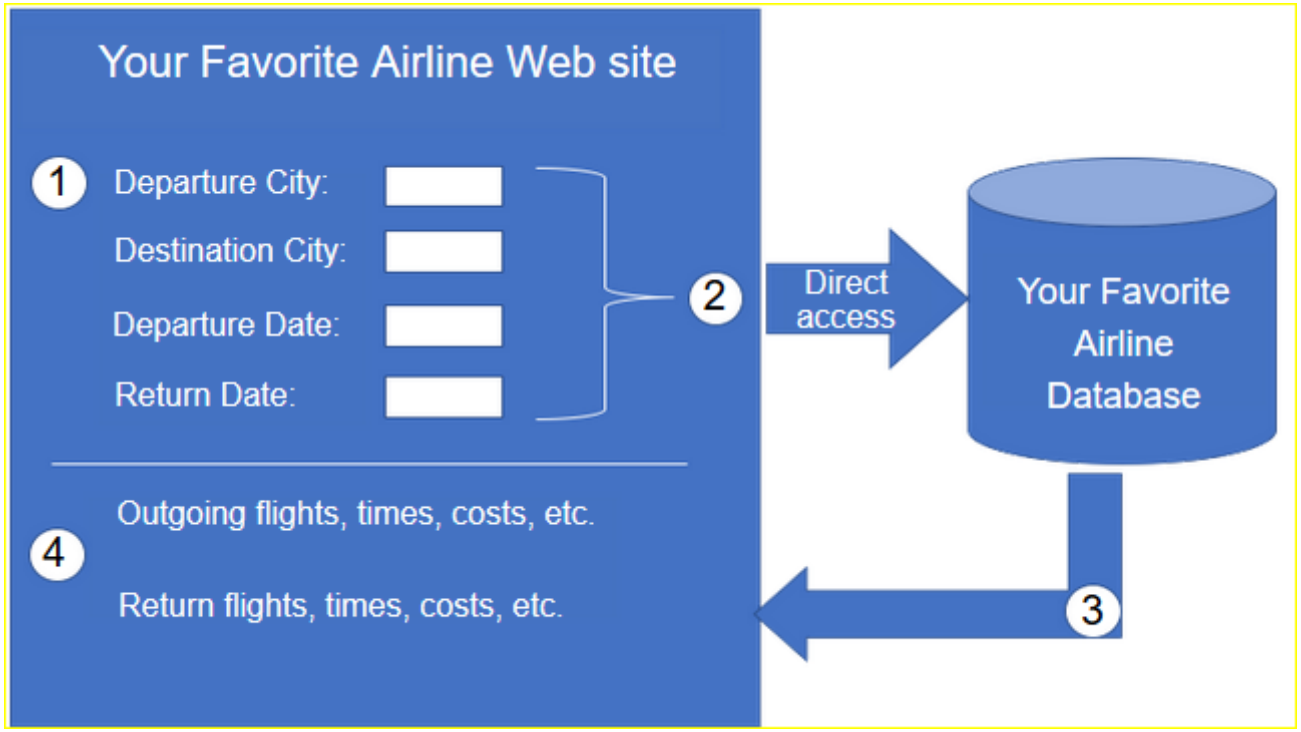
API похож на официанта в ресторане, как показано на следующем рисунке.



14.3.2 ПРИМЕР API

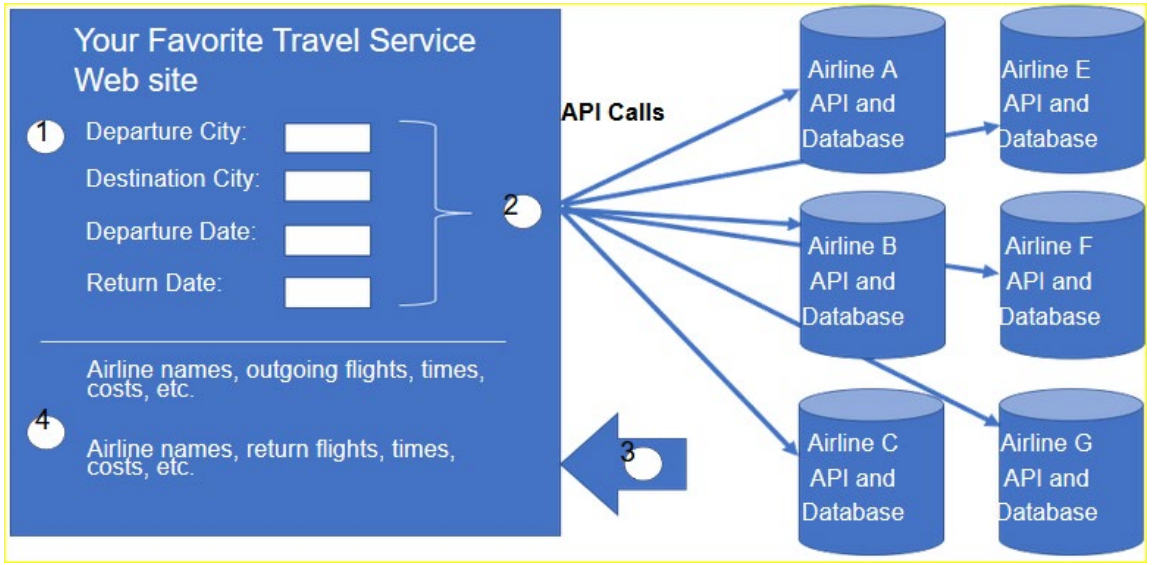
Чтобы действительно понять, как можно использовать API для предоставления данных и услуг, мы рассмотрим два варианта бронирования авиабилетов.

Первый вариант использует веб-сайт конкретной авиакомпании. Используя веб-сайт авиакомпании, пользователь вводит информацию, чтобы сделать запрос на бронирование. Веб-сайт напрямую взаимодействует с собственной базой данных авиакомпании и предоставляет пользователю информацию, соответствующую запросу пользователя.



14.3.2 ПРИМЕР API

Другой вариант. Пользователи могут использовать сайт путешествий для доступа к этой же информации не только конкретной авиакомпании, но и множества авиакомпаний. В этом случае пользователь вводит аналогичную информацию о бронировании. Веб-сайт туристического сервиса взаимодействует с различными базами данных авиакомпаний, используя API, предоставляемые каждой авиакомпанией. Служба путешествий использует API каждой авиакомпании для запроса информации от этой конкретной авиакомпании, а затем отображает информацию от всех авиакомпаний на своей веб-странице.
API действует как «мессенджер» между запрашивающим приложением и приложением на сервере, который предоставляет данные или услугу. Сообщение от запрашивающего приложения серверу, на котором находятся данные, называется вызовом API.



14.3.3 ОТКРЫТЫЕ, ВНУТРЕННИЕ И ПАРТНЕРСКИЕ API

При разработке API важно учитывать различие между открытым, внутренним и партнерским API:

Открытые API или публичные API - эти API общедоступны и могут использоваться без ограничений. Поскольку эти API являются общедоступными, многие поставщики API, такие как Google Maps, требуют, чтобы пользователь получил бесплатный ключ или токен перед использованием API. Это помогает контролировать количество запросов API, которые они получают и обрабатывают.

Внутренние или частные API - это API, которые используются организацией или компанией для доступа к данным и услугам только для внутреннего использования. Примером внутреннего API является предоставление авторизованным продавцам доступа к внутренним данным о продажах на своих мобильных устройствах.

Партнерские API - это API, которые используются между компанией и ее деловыми партнерами или подрядчиками для облегчения деловых отношений между ними. Деловой партнер должен иметь лицензию или другую форму разрешения на использование API. Туристический сервис, использующий API авиакомпании, является примером партнерского API.

14.3.4 ТИПЫ ВЕБ-СЕРВИСОВ API

Веб-сервис - это сервис, доступный через Интернет с использованием всемирной сети. Существует четыре типа веб-сервисов API:

Simple Object Access Protocol (SOAP)

Representational State Transfer (REST)

eXtensible Markup Language-Remote Procedure Call (XML-RPC)

JavaScript Object Notation-Remote Procedure Call (JSON-RPC)

Характеристика	SOAP	Архитектура REST	XML-RPC	JSON-RPC
Формат данных	XML	JSON, XML, YAML и другие	XML	JSON
Первый выпуск.	1998	2000	1998	2005
Сильные стороны	Хорошо зарекомендовавший себя	Гибкое форматирование, наиболее широко используемый	Хорошо зарекомендовавший себя, простота	Простота

14.4 REST
14.4.1 REST И RESTFUL API

Веб-браузеры используют HTTP или HTTPS для запроса (GET) веб-страницы. При успешном запросе (код состояния HTTP 200) веб-серверы отвечают на запросы GET веб-страницей в кодировке HTML.

Проще говоря, REST API - это API, который работает поверх протокола HTTP. Он определяет набор функций, которые разработчики могут использовать для выполнения запросов и получения ответов по протоколу HTTP, например, GET и POST.

Соответствие ограничениям архитектуры REST обычно называется «RESTful». API можно считать «RESTful», если он имеет следующие функции:

- **Клиент-Сервер (Client-Server)** - Клиент управляет интерфейсом (front end), а сервер обработкой данных (back end). Каждый может быть заменен независимо друг от друга.
- **Без сохранения состояния (Stateless)** - Никакие клиентские данные не хранятся на сервере между запросами. Состояние сессии сохраняется на клиенте.
- **Кешируемый (Cacheable)** - Клиент может кешировать запрос для улучшения производительности.

14.4.2 РЕАЛИЗАЦИЯ RESTFUL

Веб-сервисы RESTful реализованы с использованием HTTP. Это набор ресурсов с четырьмя определенными аспектами:

- Базовый унифицированный идентификатор ресурса (URI) для веб-службы, такой как http://example.com/resources.
- Формат данных, поддерживаемый веб-сервисом. Это часто JSON, YAML или XML, но это может быть любой другой формат данных, который является допустимым стандартом гипертекста.
- Набор операций, поддерживаемых веб-сервисом с использованием методов HTTP.
- API должен быть управляемым гипертекстом.

API RESTful используют общие методы HTTP, включая POST, GET, PUT, PATCH и DELETE. Как показано в следующей таблице, они соответствуют операциям RESTful: создание, чтение, обновление и удаление (Create, Read, Update, Delete или CRUD).

Метод HTTP	Операция RESTful
POST	Create (Создать)
GET	Read (Чтение)
PUT/PATCH	Update (Обновить)
DELETE	Delete (Удалить)

14.4.3 URI, URN И URL

Веб-ресурсы и веб-службы, такие как RESTful API, идентифицируются с помощью URI. URI - это строка символов, которая идентифицирует конкретный сетевой ресурс. URI обеспечивает:

- Единое имя ресурса (Uniform Resource Name - URN) - идентифицирует только пространство имен ресурса (веб-страница, документ, изображение и т. д.) без ссылки на протокол.

- **Унифицированный указатель ресурса (Uniform Resource Locator - URL)** - определяет сетевое местоположение конкретного ресурса в сети. HTTP или HTTPS URL обычно используются в веб-браузерах. Другие протоколы, такие как FTP, SFTP, SSH и другие, могут использовать URL. URL с использованием протокола SFTP может выглядеть так: sftp://sftp.example.com.

URI https://www.example.com/author/book.html#page155 состоит из следующих частей:

- Протокол/Схема - (Protocol/scheme) –HTTPS или другие протоколы, такие как FTP, SFTP, mailto и NNTP

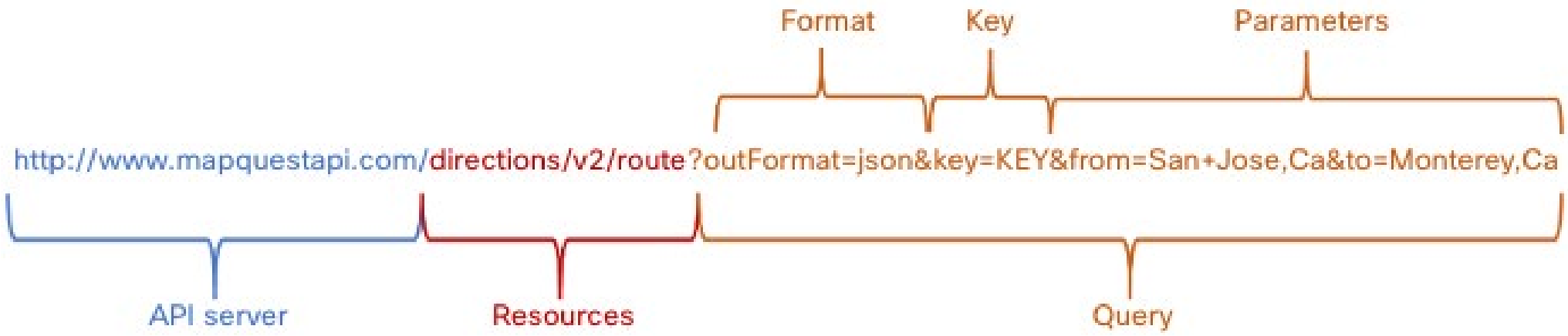
- Имя хоста - www.example.com

- Путь и имя файла - /author/book.html

- Фрагмент - #page155

14.4.4 АНАТОМИЯ RESTFUL-ЗАПРОСА

В веб-службе RESTful запрос к URI ресурса вызовет ответ. Ответом будет полезная нагрузка, обычно отформатированная в JSON, но может быть HTML, XML или каким-либо другим форматом. На рисунке показан URI для API сервиса MapQuest. Запрос API для указания направления из Сан-Хосе, штат Калифорния в Монтерей, штат Калифорния.



14.4.4 АНАТОМИЯ RESTFUL-ЗАПРОСА

Выделяют следующие части запроса API:

API server- это URL-адрес сервера, который отвечает на запросы REST. В данном примере это сервер API MapQuest.

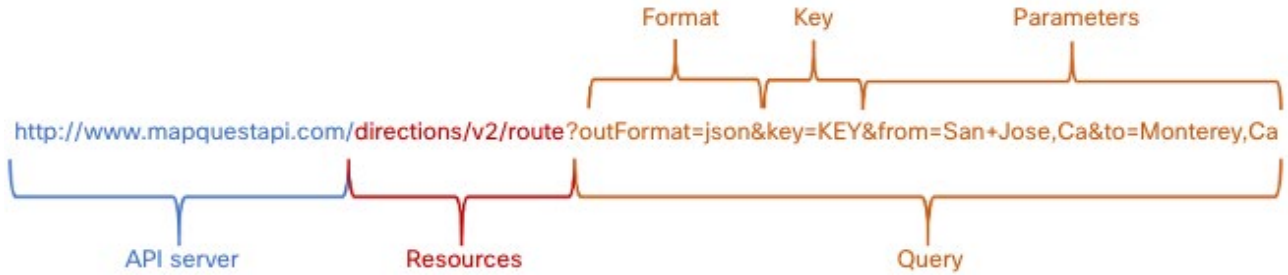
Resources - Определяет API, который запрашивается. В этом примере это API направлений MapQuest.

Query - Определяет формат данных и информацию, которую клиент запрашивает у службы API. Запросы могут включать в себя:

- Формат - Обычно это JSON, но может быть YAML или XML. В этом примере запрашивается JSON.

- Ключ - Ключ для авторизации, если требуется. MapQuest требует ключ для своих API. В приведенном выше URI вам потребуется заменить «KEY» на действительный ключ, чтобы отправить действительный запрос.

Параметры - Параметры используются для отправки информации, относящейся к запросу. В этом примере параметры запроса включают информацию о направлениях, которые нужны API, чтобы он знал, какие направления возвращать: "from=San+Jose,Ca" и "to=Monterey,Ca".



14.4.4 АНАТОМИЯ RESTFUL-ЗАПРОСА

Многие API-интерфейсы RESTful, включая общедоступные API, требуют ключа. Ключ используется для определения источника запроса. Вот несколько причин, по которым провайдеру API может потребоваться ключ:

- Для проверки подлинности источника необходимо убедиться, что он авторизован для использования API.
- Для ограничения числа людей, использующих API.
- Для ограничения количество запросов на пользователя.
- Для лучшего сбора и отслеживания данных, запрашиваемых пользователями.
- Для сбора информации о людях, использующих API.

14.4.5 ПРИЛОЖЕНИЯ RESTFUL API

Многие веб-сайты и приложения используют API для доступа к информации и предоставления услуг своим клиентам.

Некоторые запросы RESTful API можно выполнить, введя URI в веб-браузере. API направлений MapQuest является примером этого. Запрос RESTful API также может быть сделан другими способами.

Разработчики веб-сайтов: Разработчики часто поддерживают веб-сайты, которые включают информацию об API, информацию о параметрах и примеры использования. Эти сайты также могут позволять пользователю выполнять запрос API на веб-странице разработчика, вводя параметры и другую информацию.

Postman: Это приложение для тестирования и использования REST API. Он содержит все необходимое для построения и отправки запросов REST API, включая ввод параметров запроса и ключей.

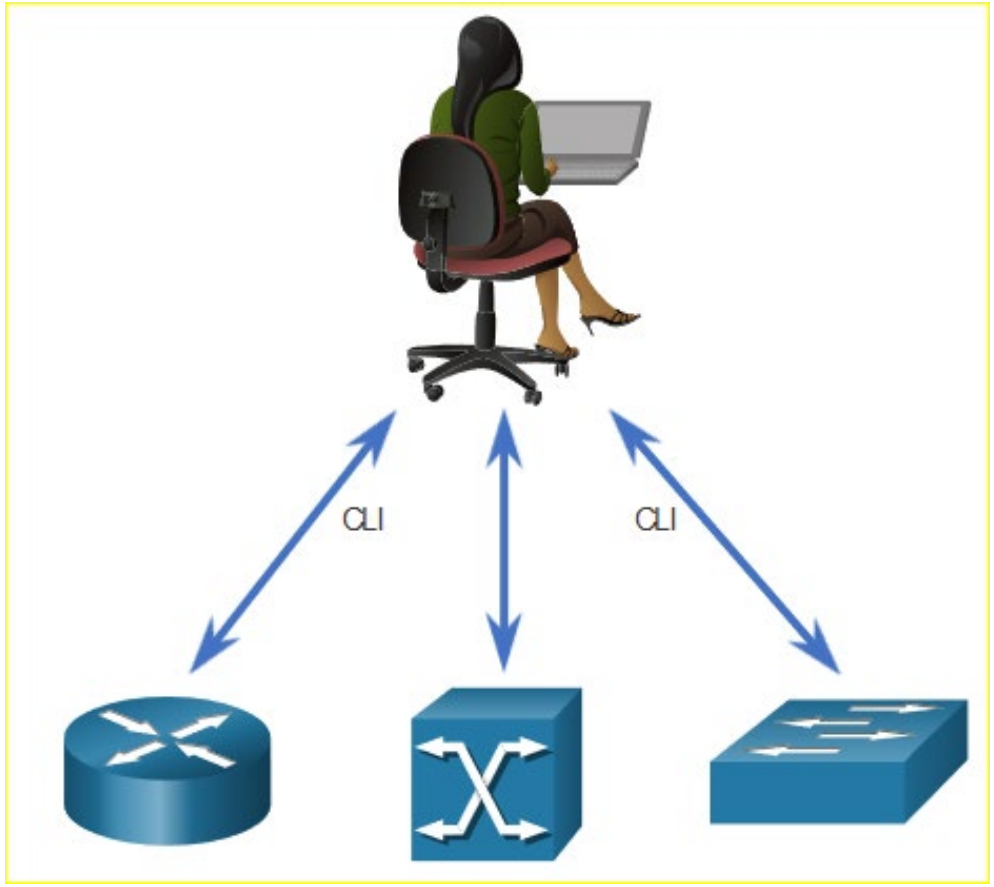
Python: API также можно вызывать из программы Python. Это учитывает возможную автоматизацию, настройку и интеграцию приложения с API.

Сетевые операционные системы: Используя такие протоколы, как NETCONF (NET CONFiguration) и RESTCONF, сетевые операционные системы начинают предоставлять альтернативный метод для конфигурации, мониторинга и управления.

14.5 ИНСТРУМЕНТЫ УПРАВЛЕНИЯ КОНФИГУРАЦИЕЙ

14.5.1 ТРАДИЦИОННАЯ КОНФИГУРАЦИЯ СЕТИ

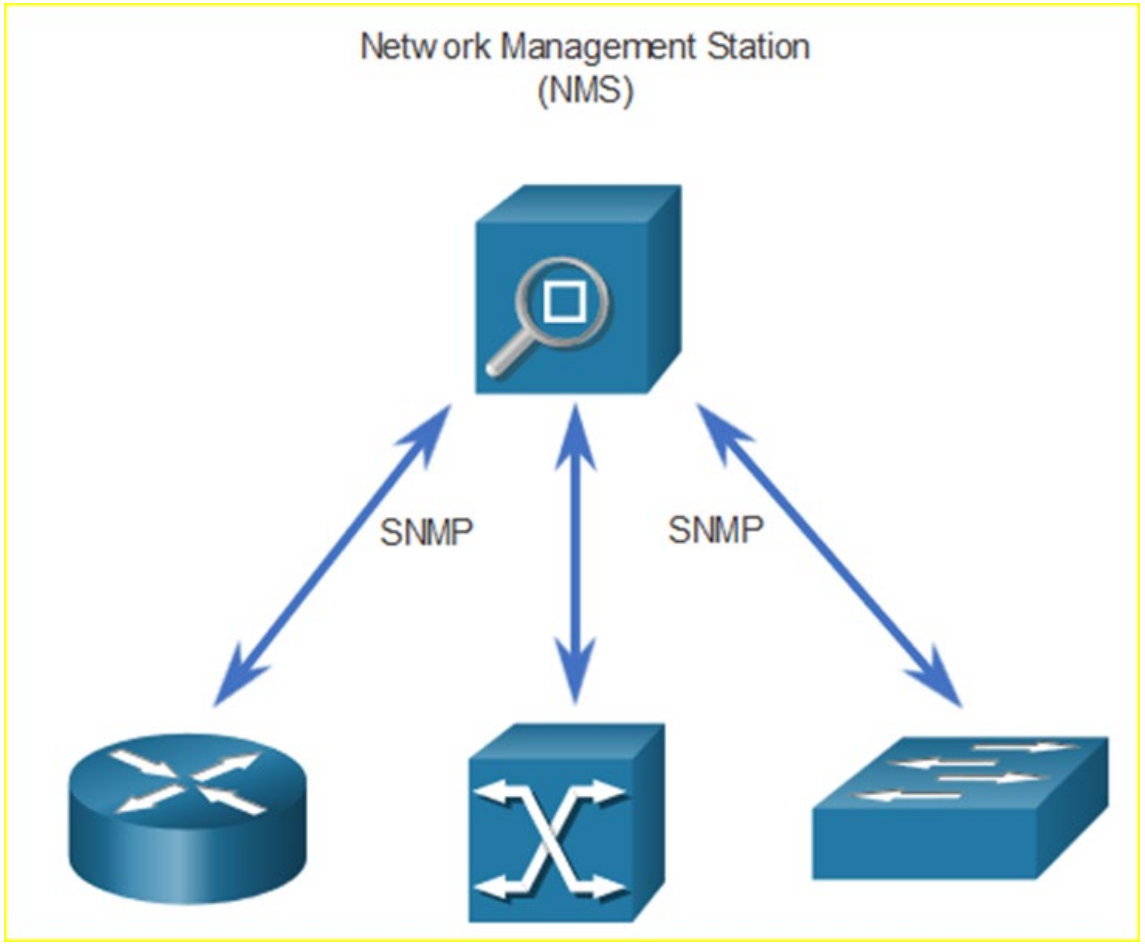
Сетевые устройства, такие как маршрутизатор, коммутаторы и брандмауэры, традиционно настраиваются сетевым администратором с помощью интерфейса командной строки, как показано на рисунке. Всякий раз, когда есть изменение или новая функция, необходимые команды конфигурации должны быть введены вручную на всех соответствующих устройствах. Это становится серьезной проблемой в больших сетях или в сетях со сложными конфигурациями.



14.5.1 ТРАДИЦИОННАЯ КОНФИГУРАЦИЯ СЕТИ

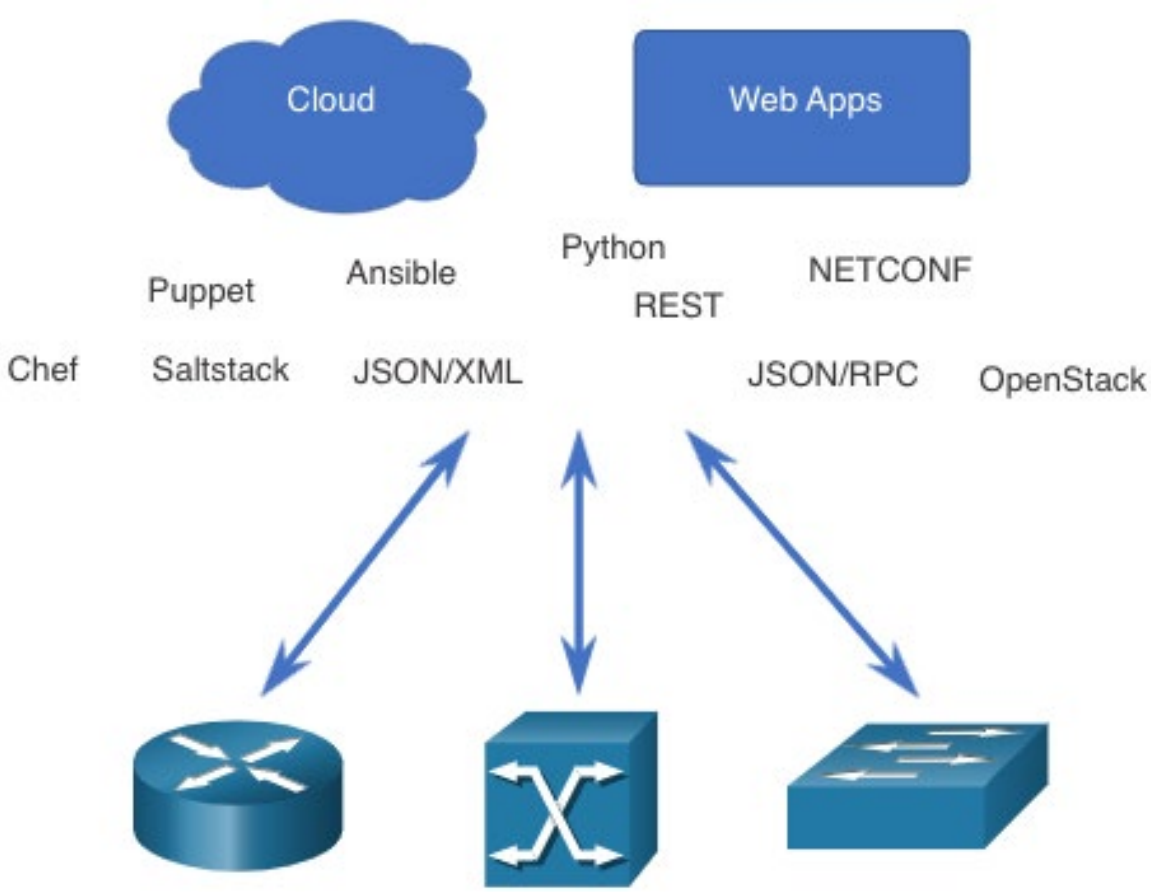
SNMP позволяет сетевым администраторам выполнять мониторинг узлов сети и управлять ими. Используя станцию управления сетью (NMS) протокол SNMP позволяет сетевым администраторам отслеживать и управлять производительностью сети, находить и решать сетевые проблемы, а также выполнять запросы для получения статистики. Но он обычно не используется для конфигурации из-за проблем безопасности и сложности в реализации.

Вы также можете использовать API-интерфейсы для автоматизации развертывания и управления сетевыми ресурсами. Вместо того чтобы администраторы сети вручную настраивал порты, списки доступа, QoS и политики балансировки нагрузки, они могут использовать инструменты для автоматизации конфигураций.



14.5.2 АВТОМАТИЗАЦИЯ СЕТИ

Мы быстро уходим от мира, в котором сетевой администратор управляет несколькими десятками сетевых устройств, к миру, в котором они разворачивают и управляют сотнями, тысячами и даже десятками тысяч сложных сетевых устройств (физических и виртуальных) с помощью программного обеспечения. Это преобразование быстро распространяется начиная с центра обработки данных и заканчивая всей сетью. Для операторов сети существуют новые различные методы для автоматического мониторинга, управления и настройки сети. К ним относятся такие протоколы и технологии, как REST, Ansible, Puppet, Chef, Python, JSON, XML и другие.



14.5.3 ИНСТРУМЕНТЫ УПРАВЛЕНИЯ КОНФИГУРАЦИЕЙ

Существует несколько инструментов, облегчающих управление конфигурацией:

- Ansible
- Chef
- Puppet
- Saltstack

Цель всех этих инструментов - уменьшить сложность и сэкономить время, необходимое для настройки и поддержки крупной сетевой инфраструктуры с сотнями, даже тысячами устройств. Эти же инструменты могут быть полезны и для небольших сетей.

14.5.4 СРАВНЕНИЕ ИНСТРУМЕНТОВ УПРАВЛЕНИЯ КОНФИГУРАЦИЯМИ PUPPET, CHEF, ANSIBLE И SALTSTACK

Ansible, Chef, Puppet и SaltStack поставляются с документацией по API для настройки запросов RESTful API. Все они поддерживают JSON и YAML, а также другие форматы данных. В следующей таблице приведено краткое изложение основных характеристик инструментов управления конфигурациями Ansible, Puppet, Chef и SaltStack.

Характеристика	Ansible	Chef	Puppet	Saltstack
Язык программирования	Python + YAML	Ruby	Ruby	Python
На основе агента или без использования агентов?	Без использования агентов	На основе агента	Поддерживает оба варианта	Поддерживает оба варианта
Как управляются устройства?	Любое устройство может быть «контроллером»	Chef мастер	Puppet мастер	Salt мастер
Что создано инструментом?	Сборник сценариев (Playbook)	Книга рецептов (Cookbook)	Манифест (Manifest)	Базовый компонент (Pillar)

14.6 СЕТИ НА ОСНОВЕ НАМЕРЕНИЯ И CISCO DNA CENTER

14.6.1 ОБЗОР СЕТИ НА ОСНОВЕ НАМЕРЕНИЯ

IBN (Intent-based networking)- это развивающаяся отраслевая модель для следующего поколения сетей. IBN основывается на программно-определяемой сети (Software-Defined Networking - SDN), преобразуя аппаратно-ориентированный и ручной подход к проектированию и эксплуатации сетей в программно-ориентированный и полностью автоматизированный подход.

Бизнес-цели для сети выражены как намерения. IBN фиксирует бизнес-намерения и использует аналитику, машинное обучение и автоматизацию для непрерывного и динамического выравнивания сети по мере изменения потребностей бизнеса.

IBN фиксирует и преобразует деловые намерения в сетевые политики, которые можно автоматизировать и применять последовательно в сети.

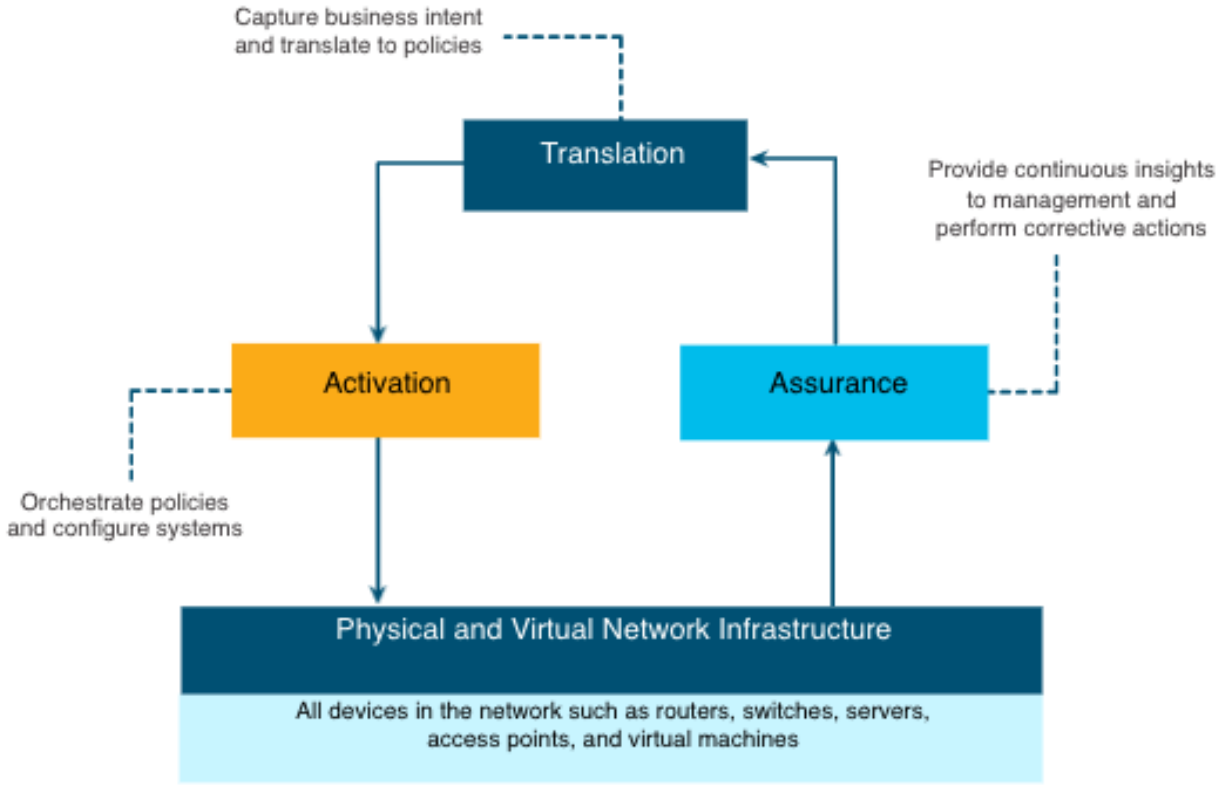
14.6.2 ОБЗОР СЕТИ НА ОСНОВЕ НАМЕРЕНИЯ

Cisco рассматривает IBN как подход, имеющий три основных функции: трансляция, активация и контроль. Эти функции взаимодействуют с базовой физической и виртуальной инфраструктурой, как показано на рисунке.

Трансляция. Эта функция позволяет сетевому администратору выразить ожидаемое сетевое поведение, которое будет наилучшим образом соответствовать деловым намерениям.

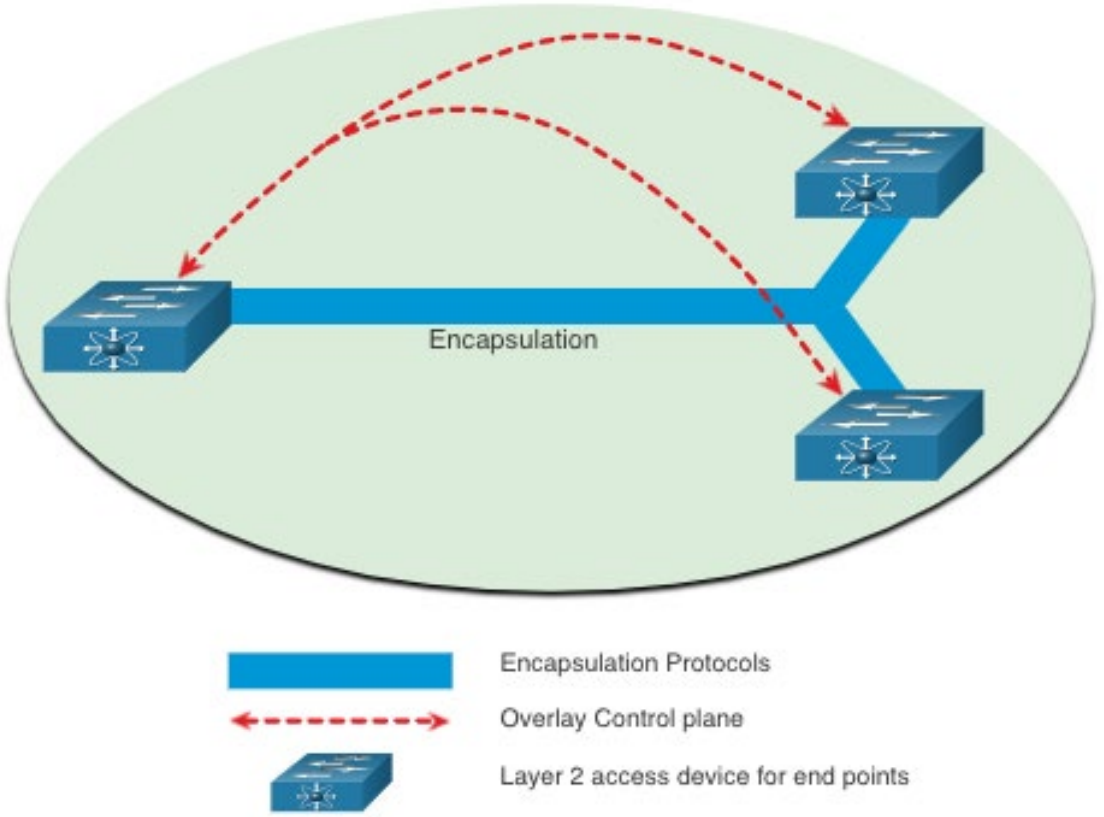
Активация. Полученное намерение затем необходимо интерпретировать в политики, которые могут быть применены к сети. Функция активации внедряет эти политики в физическую и виртуальную сетевую инфраструктуру с помощью общесетевой автоматизации.

Контроль. Чтобы удостовериться в том, что выраженное намерение реализуется в сети на постоянной основе, функция контроля выполняет непрерывный цикл проверки и верификации.



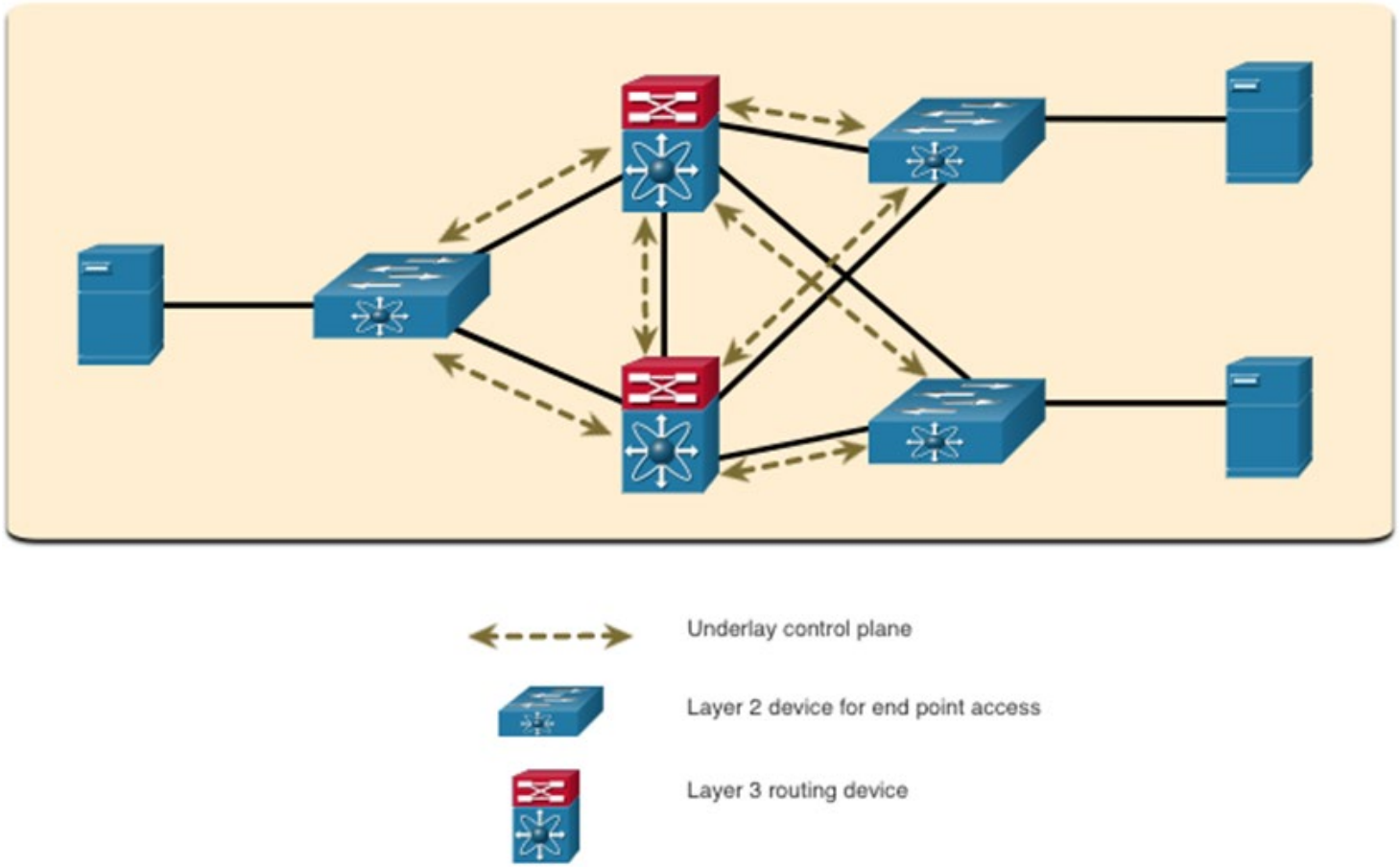
14.6.3 СЕТЕВАЯ ИНФРАСТРУКТУРА КАК СТРУКТУРА

С точки зрения IBN поверх физической и виртуальной сетевая инфраструктуры работает наложенная сетевая инфраструктура, которая представляет логическую топологию, используемую для виртуального подключения к устройствам. Такое наложение ограничивает количество устройств, которые администратор сети должен запрограммировать, и предоставляет услуги и альтернативные методы переадресации, которые не контролируются базовыми физическими устройствами. Наложенная сетевая инфраструктура - это место, где встречаются протоколы инкапсуляции, такие как IPsec и CAPWAP. Используя решение IBN, сетевой администратор может указать через политики именно то, что происходит в плоскости управления наложением. Обратите внимание, что физическое подключение коммутаторов не является проблемой наложения.



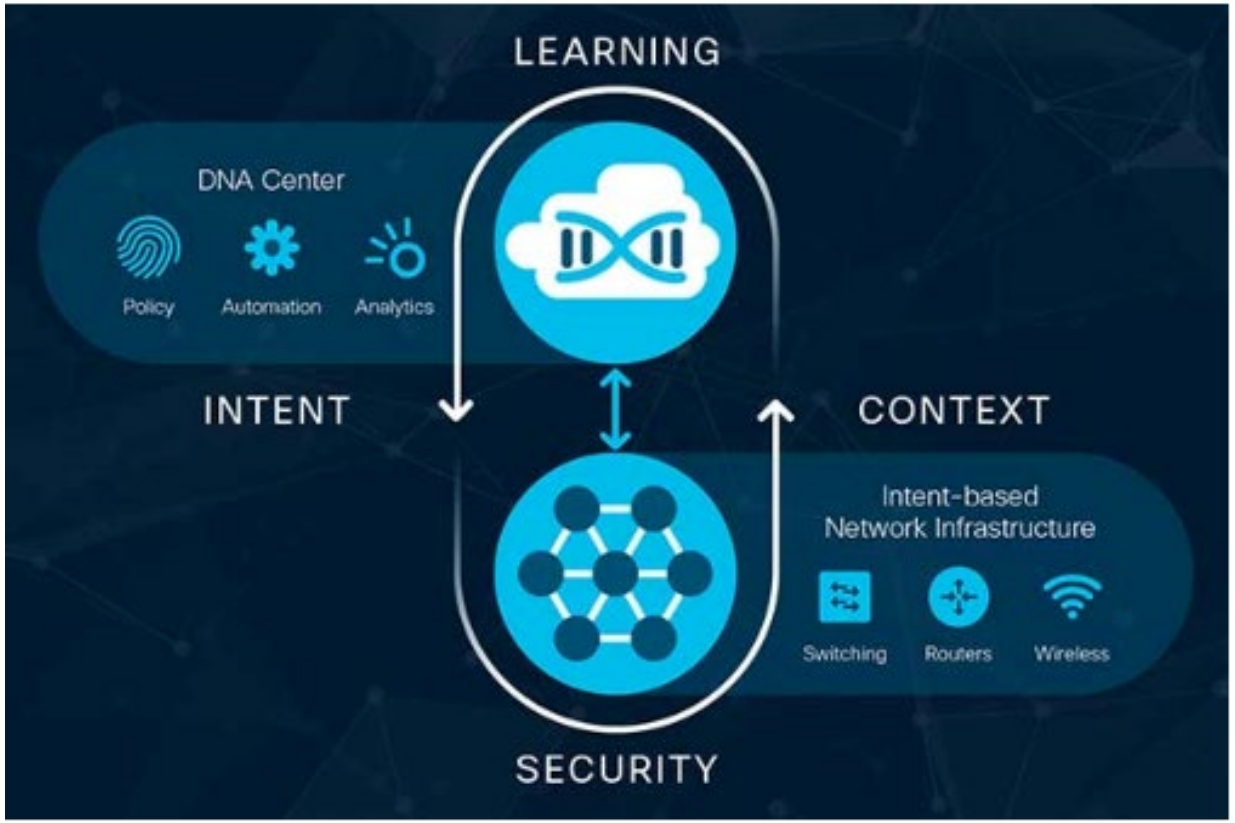
14.6.3 СЕТЕВАЯ ИНФРАСТРУКТУРА КАК СТРУКТУРА

Подложенная сеть (underlay network) - это физическая топология, которая включает в себя все оборудование, необходимое для достижения бизнес-целей. Сеть, которая определяет дополнительные устройства и указывает, как эти устройства подключены.
Конечные точки, такие как серверы на рисунке, получают доступ к сети через устройства уровня 2. Плоскость управления подложкой отвечает за простые задачи пересылки.



14.6.4 АРХИТЕКТУРА ЦИФРОВЫХ СЕТЕЙ CISCO DNA

Cisco внедряет структуру IBN, используя DNA (Digital Network Architecture). Как показано на рисунке, бизнес-намерение безопасно развертывается в сетевой инфраструктуре (фабрике). Затем Cisco DNA непрерывно собирает данные из множества источников (устройств и приложений), чтобы предоставить обширный контекст информации. Затем эту информацию можно проанализировать, чтобы убедиться, что сеть работает безопасно на своем оптимальном уровне и в соответствии с бизнес-намерениями и сетевыми политиками.



14.6.4 АРХИТЕКТУРА ЦИФРОВЫХ СЕТЕЙ CISCO DNA

Решения Cisco DNA	Описание	Преимущества
Программно-определяемый доступ	- Основанное на намерениях корпоративное сетевое решение, построенное с использованием Cisco DNA. - Используется единая сетевая структура в локальной сети и в сети WLAN для создания согласованного и надежного пользовательского интерфейса. - Сегментирует трафик пользователей, устройств и приложений и автоматизирует политики доступа пользователей, чтобы установить правильную политику для любого пользователя или устройства с любым приложением в сети.	Предоставляет доступ к сети за считанные минуты любому пользователю или устройству для любого приложения без ущерба для безопасности.
SD-WAN	- Использует безопасную облачную архитектуру для централизованного управления WAN-соединениями. - Это упрощает и ускоряет доставку безопасных, гибких и многофункциональных услуг WAN для подключения центров обработки данных, филиалов, кампусов и т.д.	- Обеспечивает удобство работы пользователей с приложениями запущенными локально и в облаке. - Достигается большая гибкость и экономия затрат за счет упрощения развертывания и независимости от транспортной инфраструктуры.

14.6.4 АРХИТЕКТУРА ЦИФРОВЫХ СЕТЕЙ CISCO DNA

Решения Cisco DNA	Описание	Преимущества
Cisco DNA Assurance	- Используется для устранения неполадок и повышения производительности IT. - Применяет расширенную аналитику и машинное обучение для повышения производительности и разрешения проблем, а также прогнозирования для гарантии производительности сети. - Делает уведомления в режиме реального времени для условий сети, которые требуют внимания.	- Позволяет идентифицировать основные причины и предлагает варианты исправлений для более быстрого устранения неполадок. - Cisco DNA Center предоставляет простую в использовании единую панель мониторинга с возможностями анализа и детализации. - Машинное обучение постоянно улучшает сетевой интеллект для прогнозирования проблем до их возникновения.
Безопасность Cisco DNA	- Обеспечивает эффективный мониторинг, используя сеть в качестве датчика для получения сведений и анализа в реальном времени. - Обеспечивает расширенный детальный контроль для реализации политики и сдерживания угроз по всей сети.	- Уменьшение риска и защита организации от угроз - даже в зашифрованном трафике. - Обеспечивает эффективный мониторинг с 360-градусным обзором благодаря аналитике в реальном времени для детального изучения. - Снижение сложности благодаря сквозной безопасности.

14.6.5 CISCO DNA CENTER

Cisco DNA Center - это базовый контроллер и аналитическая платформа, на которых основано решение Cisco DNA. Он поддерживает выражение намерения для нескольких вариантов использования, включая базовые возможности автоматизации, предоставление структуры и сегментацию на основе политик в сети предприятия. Cisco DNA Center - это центр управления сетями и командный центр для подготовки и настройки сетевых устройств. Это аппаратная и программная платформа, обеспечивающая «единый интерфейс», ориентированный на обеспечение достоверности, аналитику и автоматизацию.

На стартовой странице интерфейса DNA Center содержится общая сводная информация о состоянии и снимок сети. Отсюда администратор сети может быстро перейти к интересующим областям.

14.6.5 CISCO DNA CENTER

Вверху меню обеспечивает доступ к пяти основным областям DNA-центра. Как показано на рисунке, это:

Design - Моделирует всю сеть, от сайтов и зданий до устройств и каналов связи, как физических, так и виртуальных, в пределах кампуса, филиала, глобальной сети и облака.

Policy - Используйте политики для автоматизации и упрощения управления сетью, снижения затрат и рисков, а также ускорения развертывания новых и улучшенных услуг.

Provision - Предоставляйте пользователям новые услуги с легкостью, скоростью и безопасностью в корпоративной сети, независимо от размера и сложности сети.

Assurance - Используйте упреждающий мониторинг и информацию из сети, устройств и приложений, чтобы быстрее прогнозировать проблемы и гарантировать, что изменения политики и конфигурации соответствуют бизнес-целям и пожеланиям пользователей.

Platform - Используйте API-интерфейсы для интеграции с предпочитаемыми вами IT-системами, чтобы создавать комплексные решения и добавлять поддержку устройств разных производителей.