Introduction to Java

Without knowing it, you've probably interacted with CICS already today. Perhaps you checked your bank balance or bought a bagel for breakfast? CICS is used by enterprises to provide the services that the world depends on, but you may have never even heard of it before.

This course introduces CICS, an unparalleled mixed language application server. We explore what an application server is, what it provides to developers and what distinguishes CICS as a mixed language application server. Through the use of a illustrative application, we explain how CICS can provide shared programming contexts such as reliability, scalability, security and monitoring to applications written in a variety of programming languages - from Java Enterprise Edition to COBOL.

Section 1 Lecture 1

Hi and welcome to the IBM Redbooks video education course, Introduction to CICS. My name is Will Yates and I'm part of the CICS development team, based in the IBM Hursley lab in the UK. This course is aimed at anyone who wants to understand what CICS is and the role it plays in the enterprise. This may include business leaders, technical engineers or students of computing and enterprise systems. A very basic understanding of computing and programming is beneficial to follow along with the course. However even without this knowledge the majority of the course will still be understandable.

In this course we will give you a basic understanding of what CICS or C – I – C – S actually is, what it is used for and why businesses use CICS. This will include an explanation of the role of the application server in running applications and the benefits it provides.

We will then go on to explore the services that are provided by an application server and what additional value CICS provides in comparison to other application servers. We will then look at how those services are defined to applications running in CICS. Finally, we will discuss how CICS can scale to handle exceptionally high workloads and achieve the highest levels of availability and reliability. By the end of the course you should have a good understanding of what CICS is.

Thank you for watching.

Section 2 Lecture 1

In this section we will give a brief explanation of what CICS is, describe who uses it and the types of applications that they use it for. We will then describe the advantages of using CICS over other solutions. To achieve this we will introduce a sample application that we use to illustrate the concepts.

As you may have gathered, CICS is an application server which runs on IBM z Systems which is also known as the mainframe. We will explain a lot more about what an application server is in the next lecture, but basically it is software that is used to host applications. Most application servers tend to support applications written in a single programming language, such as java. However no single programming language is optimal for every application requirement. CICS on the other hand, can host applications written in different programming languages. this capability makes CICS an un-paralleled mixed language application server.

CICS supports modern programming languages, with the most commonly used being Java, where we have support for Java Enterprise Edition Full Platform, as well as more traditional enterprise programming languages such as COBOL, or PL ONE. Programs written in these more traditional programming languages still make up a large proportion of enterprise applications. We also support many other languages such as C, PHP or even third party languages from other vendors. This support for a variety of languages has allowed developers to write a range of applications to support their business.

CICS is used as an application server by a host of different companies in numerous industries, such as Banking, Transportation, Retail and insurance. In fact the majority of Fortune 500 companies will have CICS as part of their IT infrastructure. Typically businesses that use CICS are larger organizations who require applications that can deal with vast amounts of data and high throughput workloads. These applications often form the core of their business and are vital to their success.

Because of the importance of these applications to the business, if they were to fail, this unexpected downtime could result in hundreds of thousands of dollars in lost revenue. This means that they need a highly available, reliable, robust platform to run their applications. Examples of these type of applications are stock trading applications, or credit card processing.

And this is where CICS excels, as it is designed to handle these high intensity, high value applications. As an example It is estimated that - on average – google processes around forty to sixty thousand searches per second. In comparison performance studies of CICS have achieved transactions-per-second rates of over two-hundred thousand, proving CICS' capability in these high workload scenarios.

Whilst high workload capability is a key consideration for these applications, there are other core qualities of service which are essential – and which CICS provides. Firstly, and linked to workload, is scalability. An application server needs to be able to scale to handle both quiet and busier periods of work whilst maintaining cost efficiency and responsiveness.

The application server needs to provide high levels of reliability, to give the expected result to any request. CICS provides numerous facilities to ensure reliability, such as being able to route around known errors and to recover in-flight work in the event of a failure.

Then finally as we mentioned with regards to the impact of unexpected downtime, it must provide extremely high levels of availability. Often the demanded requirement for essential systems is quoted as the 'five nines' namely ninety-nine point nine-nine-nine percent available. We have users of CICS who have not experienced an unexpected outage in over a decade.

The qualities of service, Scalability, Reliability and Availability ensure CICS can be trusted to host those mission critical applications but those qualities of service apply equally to all types of applications. CICS, as a mixed language application server, provides a versatile application hosting environment. Where developers can write in the programming language that makes sense both in terms of their skill set and the application's purpose.

As we have mentioned, CICS on z Systems provides a very scalable environment. Meaning new applications and new workload can be introduced – without the need for additional hardware.

One of our IBM Redbooks publications focusses on this very capability, where Walmart used CICS to create applications such as a caching service which could be consumed as a cloud application, and it was this versatility and quality of service that made CICS an ideal choice to provide this service.

Enterprises and industries have trusted CICS for a number of decades, and that longevity is part of the reason why CICS continues to be trusted and utilized. It has benefited from continued evolution, refinement and incorporation of emerging technologies – ensuring it's status as the most widely used application server on z Systems.

Thank you for watching

Section 2 Lecture 2

Describing CICS as a mixed language application server only makes sense if you understand what an application server is. Previously we said that an application server is software that is used to host applications. In this lecture, we will discuss in more detail what an application server is by describing an example application and how that uses an application server to solve some common development concerns.

The easiest way to illustrate this is through an example. So let's assume the role of a travel company who wants to provide customers with the ability to book a holiday through an application, named 'book a holiday'

At its most basic, we could break down booking a holiday to the combination of booking a flight and booking a hotel. So lets look at that from an application perspective.

The first part of the application will take the customers holiday requirements and use this information to book the individual parts of the holiday. We will call this the Book Holiday program and as it is the part of the application a user will interact with, we consider it interface logic

The book holiday program will call subsequent programs which perform actions based on the information provided by the user, we consider these actions the 'business logic' in the application. Firstly, Book Holiday will call a book flight program which will read and update a central database that contains the availability of all flights, and thereby book the flight.

Then Book hotel is called to book the preferred hotel.

Once those parts have been completed, the book holiday program responds to to the user with the confirmation details of their holiday.

But suppose something goes wrong. Say that when Book Hotel was called, the hotel specified to Book Holiday was fully booked and unavailable. Or what happens if during booking a hotel the power were to fail. No user of the application wants to have booked expensive flights without a hotel to stay in, therefore the application needs to handle this situation. All the programs in the application have to be able to successfully book their components or all of the bookings need to be cancelled. This is known as atomicity which is an element of transactionality and depending on the complexity of the application, can require a lot of additional thought and coding from the developers.

The way a customer is likely to provide information to the Book a Holiday application, is through a website booking form.

This means the Book Holiday program, which calls subsequent programs, now needs to understand web communication.

Some programming languages are better suited than others to handle this communication, but any solution created is going to require significant testing effort to ensure it is reliable.

This application will be accessible by many customers at the same time. Developers have run their application under a simulated representative workload, however during this testing, different parts of the application have started to fail. These types of problems can be intermittent and therefore difficult to diagnose and fix.

If the application is going to be accessed by multiple users over the web, another obvious concern is security. How will the application be able to ascertain the identity of those using the application? And how will requests flowing between the application and the user be kept secure so they cannot be intercepted by a malicious third party?

When writing an application, developers may have to solve a number of concerns: security, transactionality, web based communication, high workloads, amongst others. These problems can be considered programming contexts which are common for most applications. However developing solutions for these issues is difficult, and takes time away from focusing on the business logic of the application.

An application server, like CICS, can solve a lot of these problems by providing services that implement these programming contexts in a generic manner. Because the application server has provided these services for the application to use, development of an application is a lot faster - as your developers can focus on the business logic of the application.

One of the ways that these services are made available by the application server is through an Application Programming Interface or API. This is simply a way of the program instructing the application server to carry out a service on its behalf. Many application servers use an API to allow applications to access services that it provides. CICS is no different as it also offers an API to be used by the programs that it hosts.

As we mentioned in the previous lecture, most application servers tend to support a single programming language and that no single programming language is optimal for every application requirement. As business requirements change, applications may need to evolve over time to provide additional services – requiring existing programs to be extended. This often leads to applications which are composed from programs written in different programming languages. For example, the Book Holiday application could be entirely written in Java, or COBOL or C, but equally the different parts may be written in any mixture of languages. In this example lets assume that book holiday is written in java, book flight is in C and book hotel is in COBOL. Linking between these programs written in mixed languages, and passing data between these programs can be particularly complicated. Whilst application servers may be able to provide services to handle the other concerns in isolation, they become far more complex when mixed languages are introduced. CICS is the application server that provides support for this mixed language interoperability.

In this lecture we have shown how writing a application is more than just writing business logic code. Handling sometime complex concerns such as communication, workload, security are difficult to implement in a highly robust manner. An application server like CICS provides a platform where applications can run and can offload some of these concerns to the application server. Using an application server can allow your application to be developed more quickly, as your application developers can focus on the business logic rather than generic computing problems. CICS in particular provides these capabilities to mixed language applications, This allows you to choose the best programming language for your requirment or to extend existing applications with new programs written in a different language.

Thank you for watching

Section 3 Lecture 1

In the previous section we introduced some of the general benefits of using an application server such as CICS and the reasons why using an application server provides developers with a strong resilient platform to build their applications upon.

In this section we will look in more detail at some of the features that CICS provides to applications, for example, different ways to call an application, such as over the web. Securing an application, calling other programs and passing data between them; accessing databases; monitoring the status of applications, and finally transactions. All of these features or shared contexts are available to applications regardless of the programming language they were written in.

By the end of this section you should understand some of the key features that CICS provides to applications. You should also understand that the capability to provide these features across different language runtimes is a key differentiator for CICS. Thank you for watching.

Section 3 Lecture 2

In this lecture we will describe how an application server provides mechanisms allowing your applications to be accessed in many different ways. Lets take a look at our example application again.

In the previous section we said that the book a holiday application would be accessed by users over the web. The application server is able to listen for incoming web requests and call the book holiday program to process these requests. Without the use of an application server the Book Holiday program would need to contain specific code to listen for a request itself. Writing high quality, reliable code that can listen for inbound web requests can be difficult, using an application server to do this means that the developers of the application don't need to be concerned with how the request is received, just know that their program will be called when a request is received.

Although not the case for our example some applications can be accessed via a queue. Queues can be a popular way to connect to applications especially in cases where the application can only execute during certain periods. If we only wanted to allow holidays to be booked during office hours of 9am to 5pm. Outside of these hours trades are queued in order until the market opens. At which point the application processes each request in order, the application server takes the message off the queue and the calls the required program.

Once the program has finished execution, the response data is used by the application server to construct a message and place it on the correct response queue.

The inbound web or queue request just describes how the data is sent to our application. The way that the data is formatted could be different based upon where the application is being called from. Two of the most common ways are JSON or XML

If the application needed to be accessed from a mobile device such as a phone or tablet then it is likely that the request will be in JSON format, this is a popular format as it is easy to read and process.

If the application were to be used by another companies application then it is likely that the application would be accessed over web services. Like mobile access Web Services sends a message over the web, however instead of JSON, XML is used to format the message. Web Services are still popular for business to business communication between applications and are widely used.

Without the use of an application server our book holiday program would need specific code to understand both the XML or JSON data that was sent over the web. Every time the data interface changed this specific code would require updating. If a new data format was introduced then all the programs that received data would need updating to handle the new format.

Instead of just passing the JSON or XML data directly to the program, CICS can transform the JSON or XML message into a format that the target program can consume, before calling the program. Once the program has finished executing, CICS can take the response message and transform it back into JSON or XML data before sending it back to the requester. CICS can transform the data to a format that the language can natively understand.

Because CICS can provide such data transformations on behalf of programs, it means that developers can expose their applications to new data types much quicker than perhaps would be possible with other application servers. Also, since the developer didn't need to change the source code of their application to be able to process the new data type, there is less risk involved. Finally because the services which CICS provides are accessible through multiple languages it means that existing applications – even pre-web applications - can be integrated into new technology with ease.

In this lecture we have discussed 2 different data formats and 2 different connection technologies, this is 4 different permutations that each application would have to write specific code to handle. However CICS provides all of these connection possibilities and more to programs written in all supported languages. CICS also provides data transformation technology to map new types of data structures such as JSON to structures that are more consumable by CICS programs. When a new data format is introduced then CICS can be updated to support the new format. The connection technologies supplied by CICS are highly efficient and reliable, meaning that you can connect your business critical applications to a wide range of connection technologies in confidence. Thank you for watching.

Section 3 Lecture 3

In this lecture we will look at the capabilities provided by an application server that help you to secure your application.

Security is a major concern for enterprise applications. How are you going to ensure that you know who your users are and how are you going to keep their data away from prying eyes?

Like most application servers, CICS provides security services to your applications helping them in three main areas, Identification and authentication, authorization and encryption

How will users of your application identify themselves? In our example we might have two users, Alice and Bob who are booking holidays. The Application needs to ensure that each user can securely identify themselves when using the application. This way the application can be sure that only Alice can view details of her holiday. The easiest way of doing this is with a username and password. When Alice or Bob initially interact with the application they send this identification token along with their request. The application server uses this information to identify each user and ensure that they only retrieve their data.

Of course a username and password is not the only way that a user can identify themselves, other methods exist such as client authentication SSL or using the open standard for authorization OAUTH. CICS is able to extract the identification of the user from the identity token on behalf of the application program. This means that applications hosted in CICS can handle many different types of identity token, since it is CICS that is handling the authentication and not the application programs.

In order for a users identity to be authenticated as correct, CICS uses the System Authentication Facility – or SAF of the operating system z/OS. All software running on zSystems can use the same facility to create, update and authenticate user tokens. This allows for a single point of control for identity management for all software running on the platform. If a users credentials need updating then this update can be performed once and it will take affect across the entire platform. Once the user token has been authenticated the rest of the users request will be assigned to that identity.

Now that the users identity has been authenticated CICS programs can use that identity to ensure that users only access resources that they are allowed to use. In our example both Bob and Alice are users of the application and as such are allowed to read and update the content of the databases that hold hotel and flight booking information. However they are not authorized to delete content in those databases, only users in the administration group have that access. CICS ensures that whenever an application access a resource like a database that the identity associated with the request has the required authority to do so.

Similar to the authentication information, the authorization rules are held in the system authentication facility and are shared across the entire platform. The rules can be applied to individual users or to groups of users, this makes administering the authorization rules as fine or coarse grained as required. The application programs often have not had to provide specific code to implement these security checks. CICS has performed the check on behalf of the program. This means that security changes can be implemented without changing the application code and also that an application program cannot skip certain security checks. For the business this means that your application runs in a very secure container.

In the same way that CICS provides capabilities for an application to be accessed, it also provides mechanisms for that access route to be encrypted. Data flowing between the user and CICS is encrypted using industry standard algorithms to ensure that if anyone were to intercept the communication they would not be able to make sense of that data. Again because CICS is supplying the capability it is available for all programs irrespective of the language that they are programmed in.

CICS provides a set of security capabilities to identify, authenticate and authorize all users of your application. Because CICS supplies all of these functions, application programs do not need to implement the checks themselves but can run assured that CICS is performing all of the security checks. CICS also supplies the ability to encrypt the data that flows between CICS and the end user. As security requirements change, so can your security implementation, often without the need to change the program source code. Since CICS also shares the SAF with other applications on zSystems then it also provides a single point of control for security across the entire platform. Thank you for watching

Section 3 Lecture 4

In this lecture we will look at how programs within an application can call each other and also pass data between themselves

As discussed previously, in our example application the book holiday program needs to call the book flight and book hotel programs. Because these programs are written in different programming languages, transferring execution between them is very difficult without the mixed language capabilities of CICS.

If all the programs were written in the same language, like Java, this is easy, all languages have ways to invoke another program written in the same language. In reality it is likely that an application would contain programs in a multitude of languages due to different tasks being better served by particular languages, a developer's personal skill set, or even newer programs calling existing programs which were written previously

In our example linking from Java to COBOL is a difficult task. Low level technologies such as JNI exist but using these is quite complex and often requires different code to be used depending on the language that the source and target programs are written in.

Another option would be to expose each program as an application in its own right. We could then use the technologies described previously to call each program using JSON or XML over the web or via a queue. This would mean that each program would run in it's own language specific application server. Although this is possible it requires the use of two additional application servers each specialising in a different language. Maintenance and administration of three different application servers is going to be more complex than with a single server. Most importantly this is a very inefficient solution to the problem as each request to book holiday would require two external service calls which is likely to increase the time it takes to process a users request. Ideally we would like a solution to allow us call programs written in different languages within the same application server. Monitoring or administering the entire application would require the user to handle 3 different application servers, each using a different monitoring or administration techniques.

CICS solves this problem by providing an API that can be used by any program in any language to call any other program. The CICS API is tailored for each language that CICS supports so that it is familiar to an application developer, however an API call in one language will have the same functional effect as the same API call in a different language. In this case the CICS LINK API allows a programmer to tell CICS to start running another program. CICS then handles the complexity of this mixed language call on behalf of the application program.

In our example the CICS LINK API is used to allow Book Holiday to call both the Book flight and book hotel programs. Since Book Holiday is written in java, the java version of the API is used. If Book flight needed to call book hotel directly then it would use a version of the API more tailored to that language. However CICS treats all of the API calls equally.

Just calling another program isn't enough, Book Holiday will also need to pass data to book flight and book hotel to instruct them of the details of the customers request, such as the intended dates of the holiday, preferred airports, hotel rating and duration

Again considering that these programs are written in different languages this is a non trivial concern, different languages represent data in distinct ways which can be incompatible with other languages. CICS provides a mechanism that allows programs to pass data called channels and containers. Each program can use this mechanism to share data with other programs.

Programs can use channels and containers to pass data. A container is simply a piece of storage that a program can use to store data that it wants to pass to another program. Each container has a unique name which can be used to reference the container by either program. A container can be as large or as small as is required. A channel is a collection of these containers, llike a container a channel has a name that can be used to reference it. CICS provides APIs to allow a program to add and remove containers of data to and from the channel. The CICS LINK API also provides a parameter to allow a program to specify the name of the channel that should be passed to the target program

For example the Book Holiday program can build a channel containing a set of containers. Each container holds a specific piece of data that book holiday needs to send to book flight. Once the program has added all of the containers that are needed it can use the LINK API call specifying the CHANNEL parameter to tell CICS to pass the channel to book flight. Book Flight can read any of the containers on the channel, update the content of a container and add new containers to the channel. When book flight completes, book holiday can access the channel again to retrieve the response from book flight.

CICS not only supports hosting programs written in multiple languages but also provides an API to allow those programs to call each other and pass data between them. This is the key function that makes CICS a mixed language application server. This makes CICS incredibly flexible as new programs can call call existing programs and re-use their logic without having to expose each program as a separate service. This solution is more maintainable and efficient than the alternative. Thank you for watching

Section 3 Lecture 5

Not all of the data our application requires will come from the user, It will need to update data that is held in databases.

In particular the book flight and book holiday programs need to access a database to store the details of hotels and flights that are available and log the details of holidays that have been booked.

Although CICS isn't a database it does allow your programs to access databases in the same way that they would if they weren't running in CICS. If your application is using DB2 also running on z/OS then you can make use of the highly efficient cross memory CICS DB2 connector.

We mentioned earlier that applications servers can handle abnormal situations. For example if book flight is able to correctly book the requested flight, however during the booking of the hotel an abnormal situation were to occur, CICS can reverse or backout the changes to the database made by the book flight program. This is because the database is treated as a recoverable resource. No specific code changes were required in the program to make use of this capability.

It's a short lecture and that's because although it's important to know that CICS supports access to databases, there isn't really a lot to discuss. The programs you write in CICS can access data held in databases in the same way they would if they were running on their own. Thank you for watching

Section 3 Lecture 6

Once an application is operational it is important to be able to monitor the operational characteristics of the application. In this lecture we will explain how CICS can help you understand if your application is meeting business level objectives and also ensure that it can meet demand.

From the perspective of the business we might have to adhere to a set of Service Level Agreements or SLAs that will define the expected performance standards for our application. Metrics such as average or maximum response time define how quickly our application should respond to users. Not only does our application have to meet these SLAs but often we will need to provide proof that we have met the SLA.

From a more technical perspective we need to understand how many times our application is used and how much of the underlying computer resource this is taking up. This will help us to plan capacity for our application during busy periods.

Application monitoring is a function provided by an application server. However this is usually for programs written in a single programming language. As we've already discussed as application change over time it may require new components to be written in another programming language. CICS has the capability to monitor the programs and resources that make up and application in a programming language neutral way. During execution of the application CICS writes monitoring records to a central location on the zSystems platform, where it is combined with other software that is running on the same platform.

This means that the data can be used to understand the number of requests to your application across the course of a day. This allows you to not only understand the total number of users but also the shape of the workload. Are there particular peaks of traffic at certain points of the day. During these periods does the response time of your application increase because of the extra workload? This might highlight a time of day where your application breaks the SLA.

As well as monitoring the entire application, CICS can provide details on each specific part of your application. For example how much of your workload is being generated from mobile data requiring transformation from JSON. It also allows you to highlight if any of the programs within your application are performing slower than the rest of your application

Since the monitoring records are written by all the applications running on zSystems it means that since our application uses as database on zSystems we can monitor that database and our applications interactions with it from a single place.

CICS provides a lot of functions to allow you to measure the performance and operational details of your application. This is of vital importance when running mission critical applications that need to have the highest levels of reliability and performance. CICS not only provides this capability across a mixed language application but also without needing the application to make any code changes. These metrics can be combined with other software running on the platform allowing you to monitor the system as a whole and not as a collection of disparate pieces. Thank you for watching

Section 3 Lecture 7

In this lecture we will look at transactions and transactionality and explain why they are important to applications

On of the main principles of transactionality is known as atomicity. Simply put this means that all operations within the transaction must succeed or none should. Lets apply this back to our sample application. As we mentioned earlier in the course, booking a holiday in our application is made of two parts. Booking the flights and the hotel. Both of the bookings must succeed or fail

Lets assume we are processing a request for a user. As soon as CICS receives the request it begins a transaction. This transaction defines the first program to run which in our case is Book Holiday and also starts a new unit of work. Book Holiday calls Book Flight to reserve flight details. Book Flight succeeds in booking the flight. The updates that Book Flight makes to the database are recorded within the unit of work. Book holiday then calls book hotel to book the rooms for the user. But there has been a problem, the hotel is full. The user might want to rebook to a different date or destination but the flights have already been reserved. In this instance CICS can be instructed to

rollback the transaction. To do this CICS uses the details within the unit of work to effectively cancel the bookings made by book flight and report the problem back to the user.

Although in our example we have show a very simple example where only two updates are made, however a unit of work within a transaction can contain many updates to different resources. Each time CICS starts to process a request it is contained within a unit of work in a transaction. All of the updates that are made as part of that transaction are kept 'in-doubt' until CICS has finished processing the request. Once complete all the updates are finalised at the same time. This implicit way of handling transactions and units of work requires no code changes and is transparent to the application programs. However if the program does wish to have more control over the transaction model then they can use two CICS API calls: syncpoint – to finalize the in-doubt unit of work and start a new one. The second is rollback - to signify that this unit of work can no longer complete and should be backed out.

Transactions are not unique to CICS, other application servers also provide transaction capability to hosted applications, For example Java Transaction Architecture or JTA is part of the Java EE full platform specification. however unlike other transaction architectures CICS can maintain a transaction over the execution of programs written in different languages. This is important as it means that even as applications are extended by new programs, the transactionality of the application is still maintained.

As we can see being able to handle transactions is an important feature of an application server. Imagine if your bank didn't handle transactions, you would be in danger of losing money whenever you tried to transfer or withdraw funds. Within CICS, the processing of each request is handled as a separate transaction. This implicit model guarantees that even without any coding changes an application will maintain data integrity even if the power were to fail. This coupled with the ability to maintain transactionality over mixed language applications is a key benefit of using CICS over other solutions. Thank you for watching

Section 3 Lecture 8

In this section we have introduced you to some of the key capabilities of CICS as an application server.

As we have discusses CICS provides the capabilities that you would expect of any application server. Elements such as security, database access and transactions are not unique to CICS. However CICS can provide these services to applications composed of programs written in a single or a mixture of different languages. This gives you unparalleled capability to run new and extend your existing applications with new programs without having to restrict your ability to operationally manage and administer your application on a single application server. Thank you for watching.

Section 4 Lecture 1

We've looked at what an application server is, and how CICS offers various services to hosted applications. In this section we'll take a closer look at what a CICS application looks like and how it makes use of those services.

In our example application, the user provides information to the application such as preferred flights and hotel.

The application then books those constituent parts of the request and will respond to the user either with a confirmation, or in the event of an error; will rollback any successful bookings and inform the user that a problem has occurred.

CICS provides services on behalf of the core business logic programs, such as handling the web communication, security, and transactionality amongst others.

To provide these services CICS will be configured through a set of resource definitions, in particular TCPIPSERVICE, URIMAP and PROGRAM. We will explain each of these definitions in turn and how they are used to configure CICS to provide web access to our application.

Let's see how this could be implemented by looking at the first program in the application book holiday, and how CICS would enable this program to be accessed over the web.

This would all start with a user sending a request for a holiday booking over the web using HTTP.

This request would be sent to a z Systems mainframe where our application is hosted in CICS. This location is specified through a hostname and a port number. The hostname is a human-readable name that corresponds to the address of a device connected to a network. A port number just represents the specific service on the host we are trying to reach.

For the purposes of this example, let's say our mainframe could be accessed through the web address or hostname cics.ibm.com, and one of the open ports was eighty. For simplicity we could say that 'AliceInfo' represents a request from Alice to list the details of her holiday.

For our application to receive this request, CICS will need to be listening on the port the request comes into, which in our example was 80.

To do this we define a TCPIPSERVICE resource in CICS. This instructs CICS to listen to a particular port number for incoming request. As well as the port number to listen on you can also specify other attributes such as security options on the connection – such as to only accept SSL encrypted connections or to require a username and password as we discussed within authentication in our security model. Once defined and installed, CICS will now open and listen to that port for incoming requests

Many applications could be listening to the same port, so although CICS has the request from the web it now has to route the request to the correct application to be processed. This routing is done by configuring another resource definition called a URI MAP. This resource allows CICS to map particular URI to the correct destination – like our program in CICS.

URIMAPs can be very granular and perform 1 to 1 mappings between a specific URI and a program, or you can use wildcards to route multiple requests to the same location. Using our example request, the URIMAP could dictate that any URI with /Holiday should be routed to the Book Holiday program. Note that all of this behaviour has been handled outside of the applications source code. The application developer has not had to implement this logic within their source code.

The URIMAP resource definition doesn't route the request directly to the compiled executable. Instead it references another resource definition that represents that compiled program. This definition is called the program resource definition and it allows us to tell CICS how and where the program should run. In our example Book Holiday is a Java Program so it will need to run within a JVM. This resource is also how other programs can reference and interact with our program such as via the LINK API we have described in previous lectures.

So far, a user has sent a request to book a holiday over the web. CICS has listened on a port for this request, accepted and read the request and routed that request to the appropriate program.

Now that the users request has reached our application, it can be processed and an appropriate response can sent back to the user using the same mechanisms as the request came in over.

The way CICS has been configured to provide this service to the application is through the use of various resource definitions, which are essentially an abstraction of the underlying resources needed by an application. A resource definition can refer to application components, such as a program written by a developer, or to aspects of the infrastructure, such as technologies which provide an HTTP connection into an application.

These definitions allow for the configuration and utilization of those underlying resources provided in CICS or the platform it is running on.

The source code of a developer's program refers to these definitions, rather than the resource itself. Meaning resource configuration can be taken care of outside of the program.

That means that if a change to these resources is required – for example if the location of a database that the program accessed information from changed – the developer would not need to change the source code and re-deploy the application, only the resource definition for the connection would need to be changed.

This capability can be particularly useful when companies want to gradually modernise or change parts of an application.

In our example application, we could swap the Book Hotel COBOL program for an equivalent in Java.

Other than the code for the Book Hotel program itself, the only thing we would need to change in this application is the corresponding program resource definition. The other components of the application could remain as they were.

This means less change, less risk and less time to deliver the changes and realise their business value.

Many different resources and programs have been used to process this request. One of the crucial services we said CICS provided was transactionality, and that if something were to go wrong we would be able to back out any changes and return to our starting state.

This means that we need to be able to group all of the actions of a specific request – from origination to completion. CICS treats all changes that should occur as part of a request as a single Unit of Work, where either these should all happen, or none should happen.

CICS also allows for a developer to use the CICS API to programmatically say when a Unit of Work should complete.

In this lecture we have shown the resource definitions that CICS would require run our application. This has included a TCPIPService, URIMAP, Program. Through these resources we can configure a program written in any language to connect to the web. There are other resource definitions that CICS provides which enable other capabilities. However the reasons behind the use of these definitions remains the same. It allows for programs to access resources in a loosly coupled way allowing the underlying implementation to change without requiring changes to the application developers source code.

Finally we discussed that nothing runs in CICS without a transaction. This transaction ensures that in the event of a failure all of the updates made by a program are backed out. Thank you for watching

Section 4 Lecture 2

Resource definitions not only configure CICS to run an application and provide services to that application. But they also act as a mechanism to control and administer your application.

Before any definitions can be used they must be defined. Definitions can be created through a command line type environment or through a graphical interface known as the CICS explorer. These definitions can be grouped together in a logical unit. This unit of definitions can then be installed into CICS.

In our sample application we will need a program definition for each of the programs, a database definition for the local database connection that book flight and book hotel will use. We will also need to define the TCPIPService, URIMAP resource definitions that were previously described.

Once the resource definitions are installed CICS is able to use the resources and can start to run our application. The installed resources also allow us to control and administer our application.

Using the same interfaces that we used to install the application we can alter the resource definitions to change the operation of our application. For example the program resource shows how many times this program has been executed. It also allows us to reload the program into CICS if the underlying source code needed to be recompiled.

The TCPIPService allows us to disable the resource which would temporarily stop CICS from listening to the port. We can also alter the security requirements of a user attempting to use the application.

In this lecture we have further discussed the use of resource definitions in CICS to not only install our application, but also to control the application once it is installed. Thank you for watching.

Section 5 Lecture 1

CICS has been carefully engineered to allow you to run a lot of workload within a single server. However there are times when your application grows beyond the capability of a single server. In this lecture we will look at some of those scenarios and describe how CICS can scale to meet any workload demand.

So far in our example we have focused on a single application running on a single instance of the CICS application server. An instance of a CICS server is commonly known as a CICS region or simply a region. Each CICS region is capable of running not just a single application but many other applications simultaneously. For example the company could launch a new application called Book a cruise. Both applications can run on the same CICS region. Each region has an upper limit of the number of tasks that is can run concurrently. CICS has evolved to increase this limit, allowing you to run more and more workload within a single region.

Although there is an upper limit to the number of concurrent tasks a region can support, running a region at this threshold is not always best practice. Processing each request is going to require system resources that might mean that running at the maximum might cause constraints elsewhere in the system. Even if 2000 tasks is adequate now, will it be enough as your enterprise grows. Running a single region is also a single point of failure. Although CICS is incredibly reliable, a rogue

application, power loss, or network loss could cause a CICS region to become unstable and cause your application to slow down or prevent it from running at all. CICS allows you to span applications across multiple CICS regions, allowing you to scale your application to meet workload demands.

At the heart of this capability is the ability for CICS to use the same link API call that we have previously discussed to start running a program located on another remote CICS region. This ability is configured through altering the CICS program resource definition.

By default all CICS program resource definitions are configured to run the program they represent locally. If the remote system attribute is specified, this configures CICS to run the program on another CICS region. Because the configuration is in the resource definition it means that no source code changes are required to enable this capability. If the programs are using channels and containers to share data between themselves; CICS will also ship these across to the remote region. In fact the programs are agnostic about the changes, meaning they are not aware that they are running in different CICS regions.

In this lecture we have explained that no matter how big your workload, CICS is ready to handle the most demanding workloads. The basic technology that enables this is the ability to transparently split an application across multiple instances of CICS and allow programs to call each other even if they are in different regions. More CICS regions means that your application can do more in parallel. Thank you for watching.

Section 5 Lecture 2

In the last lecture we looked at how CICS allows an application to be distributed over a set of different CICS regions. However how you split your application up will be important to ensure that you use this capability optimally not just for performance but for availability.

Most applications can be split into several discrete layers, each layer providing a different capability of the application. The most general layers are interface logic and business logic. Each layer is simply a subset of the programs and resource definitions that make up your application. Interface and Business logic are just examples, some applications might define new layers such as a data access owning layer if the application has specific programs that access a database.

The interface layer controls how your application can be accessed.

This layer may be split into multiple parts if your application can be accessed over multiple mechanisms such as over the web, a mobile app or from a business partner The purpose of this layer is to process the inbound communication into a common form that is consumable by the business logic layer. For example if your application can be called via web services then this layer would be responsible for converting the XML based message to native data that the programs in the business logic layer can understand.

The business logic layer is the core of your application. Programs in this layer take requests from the interface layer and respond with the required data after performing any necessary manipulation of that data. Although the interface layer might be split for each access mechanism it is likely that they will all share the same business logic layer

There may be other layers in your applications. Maybe all your applications share a common logging or data access programs, these would be contained in their own layers as well. Some of the layers might even be shared across different applications

CICS regions used to host a particular layer are often given names such as Application Owning Region (AOR) or Web Owning Region (WOR). These names are often cause confusion for new CICS users, but they are simply terms associated with a particular layer of CICS regions to help identify the purpose of that region within a solution architecture. Each CICS region isn't aware of the 'name' of the layer that it is within. It is simply a way of naming a group of CICS regions to make them easier to refer to.

So why would we split our example application into layers? Why not just duplicate the application multiple times? A key reason is that each layer of the application is likely to have different performance characteristics. A business logic layer is likely to be shared by several interface logic layers. This means as our application workload increases; or our application supports new ways of being driven, we will probably need more capacity in the business logic layer than in the interrface layers

In our application we have two layers. Since the book holiday program handles the inbound web traffic and prepares the requests to the other programs, this is our first layer a web owning region layer. The two other programs provide our business logic so they are in the same AOR layer.

We can now create two CICS regions and run Book Holiday in one region and Book Flight and Book Hotel in another. This basic architectural change gives an instant increase in the amount of requests we can process. More importantly it creates the architecture that we can use to scale our application even further and prevent single points of failure.

For example we can duplicate our AOR CICS region to create 2 AORs, meaning that the book holiday program can link to Book Flight or Book Hotel in either of the AOR regions. Ideally we need a mechanism to intelligently decide which of the two regions should be used for each request from book holiday.

CICS contains a component known as the CICS Plex System Manager which runs within a separate CICS region known as the CICSPLEX Management Address Space, or CMAS. The CMAS monitors the health of all the CICS regions that are running. It can then use this information each time Book Holiday wants to call either book flight or book hotel and calculate which is the most efficient AOR region to route the work to.

If we were to add new AORs to our configuration then the CMAS can incorporate the new region into it’s calculations, allowing work to be spread across all of the available regions efficiently. This basic concept is the key to being able to scale CICS to support increasing workloads.

CICS contains a component known as the CICS Plex System Manager which runs within a separate CICS region known as the CICSPLEX Management Address Space, or CMAS. The CMAS monitors the health of all the CICS regions that are running. It can then use this information each time Book Holiday wants to call either book flight or book hotel and calculate which is the most efficient AOR region to route the work to.

In this lecture we have described how to split an application up into distinct layers and then host each layer in a series of duplicated CICS regions. Each layer can be independently scaled to support your workload demands. CICS can monitor the health of each of the regions and route work to the most efficient region. Thank you for watching.

Section 5 Lecture 3

So far we have looked at how you can scale a CICS application across multiple CICS regions to support an increasing workload. However running your application across multiple CICS regions can also help your application to become more resilient.

As we mentioned in the beginning of the course, because CICS is used to support businesses mission critical applications, unexpected outages can be very expensive. It is vital that CICS can keep running applications during routine software maintenance or unexpected hardware failures

For example, lets take the scenario where maintenance is being applied to a single CICS region. During this maintenance the CICS region would be unable to process new work requests. The CMAS will sense that the CICS region is unavailable and intelligently route the work requests to the second CICS region, this is done transparently to the workload and the application.

Once the maintenance to the first CICS region is complete and the CICS region is restarted the CMAS will be notified that the CICS region is available again and the CMAS will automatically start to share the workload between the two regions . Because the AOR layer was split over two regions we are able to keep the application running even though one of the regions might not be available. This is part of the technology that allows CICS to be so highly available. However the same capability isn't true for the WOR. Since that layer is only being run within a single CICS region we are not able apply maintenance to that region without disrupting the workload.

To eliminate this potential point of failure we can split the WOR layer across multiple regions like we did for the AOR layer. However both of the WOR regions require the use of the same TCPIPSERVICE. The operating system zee O S allows multiple CICS regions to listen to the same TCPIP port number at the same time. This means that when a network request is received the operating system passes it to any of the CICS regions that are listening to this port. This means that the WOR layer of our application can now be scaled across multiple regions. Our application has now been spread over 4 CICS regions.

So far we have only considered resilience by scaling the number of CICS regions within a single instance of the underlying operating system, which as we have discussed previously for CICS is zee OS. Zee OS is an advanced operating system and supports the ability to run mulitple instances of itself on a single mainframe. Each instance of zee OS is known as a logical partition or LPAR. An application can be distributed across a set of CICS regions which are distributed across different LPARs.

Lets take a basic example of this. We have recently scaled our application across 4 CICS regions. These 4 regions are running in the same LPAR. We can duplicate this configuration across two logical partitions. Even in this configuration each CICS region can communicate with all the other region, even those running on the other operating system. Work that starts in one LPAR can be routed to a region in the other LPAR. Splitting your application over multiple LPARS has similar advantages to running your application over multiple CICS systems.

If maintenance needs to be applied to either LPAR then the configurationis resilient to detect the change and route workload to the other operational LPAR. Once the maintenance is completed then work again will be spread across all the CICS regions in both LPARS.

Multiple zSystems hardware can be combined together and LPARs deployed across them all. Not only does this increase the capacity of your application by increasing the hardware that will support your application, but it also increases the reliability of the application. In this highly available

configuration CICS can ensure that your vital workload remains operational during maintenance to a single CICS region, LPAR or even physical hardware failure or power outage.

Now that the application has scaled across multiple CICS regions, z/OS Operating Systems and physical hardware there is a fear that this large configuration might be very difficult to administer. Resource definitions that are used to configure CICS will have to be deployed to all the CICS regions within the configuration. The CMAS that is used to monitor the health of all CICS regions also supports single system image technology. This allows you to manage all your CICS regions from a single interface. From this single interface changes to resource definitions or new applications can be installed and deployed across all your CICS servers in a single request. This ensures that as you scale CICS to support a growing workload your adminstration time need not increase as well.

An application running in CICS can scale across mulitple CICS regions, instances of the z/OS operating system and physical machines. This ability allows CICS applications to be incredible reliable and resilient. This scalability allows an application to keep running even if the underlying technology is upgraded, or suffers failure. It is this technology that allows CICS to have the five nines' reliability. Thank you for watching.

Section 6 Lecture 1

In this course we have described that CICS is an application server that runs on IBM zSystems otherwise known as the mainframe. It is used by a range of businesses include the majority of the Fortune 500 to run their mission critical applications. Because of the nature of the applications that CICS is used to host, reliability, scalability and reliability are of the highest importance.

CICS provides a set of services to hosted applications similar to other application servers. However CICS can provide these services to mixed language applications, this means that existing applications can be extended with components written in new languages. Irrespective of programming language choice CICS offers a single API to provide services to applications including the ability for mixed language applications to call each other and share data.

As your business needs change and your workload grows so can your CICS applications, CICS applications can be scaled across multiple CICS regions, LPARs and even physical hardware. This capability doesn't require any source code changes, reducing risk and allowing you to meet your service level objectives. Single system image capabilities ensure that as your CICS environment grows your administration time does not.

No single programming language is optimal for every application requirement. That is why IBM CICS Transaction Server has evolved to become the worlds most unparalleled mixed language application server.

CICS transaction server for zee os enables applications written in different programming languages to share core programming contexts such as security, transactionality, management and monitoring, regardless of the languages those applications are written in.

Thank you for watching.