**Topologia centralizada** -> varios computadores ligados a um nó central (Switch)

**Topologia distribuída** -> varias computadores que se ligam a vários computadores de forma distinta.

TCP -> Transmission Control Protocol -> Orientado a conexão e confiável

Software que usa padrão de projetos Layers

Rede/física (Ethernet, wifi) (MAC address) (física: Hardware e Enlace: Software) ->

Internet (IP – Internet Protocol) (Conexão entre redes) (Transferência de pacotes) ->

Transporte (TCP: Confiável, UDP (melhor esforço)) ->

Aplicação (HTTP, FTP, SMTP, POP, DNS) (Comunicação processo a processo) (número da porta)

Anatomia do IPv4:

**ClasseA**:

192 (Rede) -> Poucas redes

.168.100.230 (Host) -> muitas máquinas

Máscara de subrede (tem o mesmo tamanho de um IP) -> 255.0.0.0

**ClasseB**:

192.168 (Rede)

.100.230 (Host)

Máscara de subrede -> 255.255.0.0

**ClasseC:**

192.168.100 (Rede) -> Muitas redes

.230 (Host) -> Poucas máquinas

Máscara de subrede -> 255.255.255.0

**Conceito de Porta** -> indica qual processo vai atender a requisição

Exemplo NodeJS está na porta 8080, então para acessaá-la é preciso indicar 192.168.100.230:8080

HTTP -> Hyper Text (textos linkados) -> documentos ligados entre si. Ao navegar pela internet vamos encontrando endereços de sites, palavras sublinhadas, ícones piscando, e muitos outros atrativos que nos levam a clicar com o mouse e abrir diversas janelas, pois bem, este é o chamado efeito hipertextual no ciberespaço.

* Está na camada de aplicação
* É Stateless
* Cliente-Servdor
* Utiliza o protocolo TCP/IP
* É composto por códigos HTML, CSS, JS, mídias, JSON, etc (o servidor recebe uma requisição e retorna um código html que quando é interpretado pelo cliente carrega os demais códigos(referencias).
* Fluxo da comunicação -> o cliente informa ao navegador uma (URL + Parametros) que gera uma requisição ao IP relacionado com aquela URL, o servidor recebe a requisição, gera e retornar uma resposta que é interpretada pelo navegador e apresentada ao usuário. Requisição e respostas contêm cabeçalho e corpo.
* Métodos HTTP: **GET, POST**, PUT, DELETE, TRACE, OPTIONS, CONNECT, HEAD.
* **O desenvolvedor precisa estar atento a composição de suas páginas, pois cada nova URL do documento gerará uma nova requisição ao servidor, e como o protocolo TCP possui um processo específico que antecede a transferência dos dados, uma página que realiza muitos requests para ser carregada pode ficar lenta. Isso pode ser evitado utilizando arquivos maiores ao invés de vários pequenos arquivos.**
* pesquisar: estratégias para diminuir a carga de um site (redes de altos desempenhos Content Delivery Network.
* Get: URL + PARAM ex: https://www.google.com.br/search?q=web+moderno&hl=pt-BR -> O método GET não possui corpo e os parâmetros são passados no cabeçalho.
* Para as outras requisições, os parâmetros são passados no corpo (isso não garante a segurança das informações trafegadas), o que auxilia é o protocolo HTTPS.
* Grupos de status de mensagens:
  + 1xx – informação
  + 2xx – Sucesso
  + 3xx – redirecionamento
  + 4xx – erro no cliente
  + 5xx – erro no servidor

**Servidor Web** -> Quando o cliente digita uma URL, essa URL é tratada em servidores DNS que identificam qual endereço de IP está ligado aquela URL. Uma vez resolvido o IP, ele faz a comunicação com o servidor;

**Dentro do servidor há uma estrutura de pasta** -> imagens, JS, CSS são armazenados em pastas (servidores especializados em conteúdos estáticos). É comum que as estruturas estáticas sejam armazenadas em vários servidores externos, reduzindo assim a carga do web server que recebe as requisições.

**Web Stack** -> SO – DB – WebServer – Language => o mais usado é o acrônimo LAMP (Linux/Apache/MySQL/PHP). Xamp, Wampp são exemplos.

Dendendo da situação, um único servidor web vai utilizar várias portas HTTP-> 80 HTTPS->443

**Aplicação Server Sid**e (Back End) -> Via de regra toda a aplicação, lógica e renderização são processadas no lado do servidor. Distribuição simples: Uma vez que muda o site, todos os clientes mudam. O Servidor centraliza todas as responsabilidades. Conteúdo HTML é gerado a partir do servidor.

**Aplicações Client Side** (Front-end) -> Aplicações ricas que tem vários comportamentos dentro dessa aplicação e o acesso a servidor é mais para obter dados. É comum ter o conceito de SPA (Single Page Application), a primeira página é renderizada e toda a dinâmica HTML é gerada pelo JS do front end.

**Conteúdo estático (server Side) ->** Quando o servidor para renderizar uma página leu apenas arquivos armazenados e serviu a página através desses arquivos. Pode ser tanto um arquivo enviado via FTP ou um código que gera um arquivo estático de tempo em tempo e armazena para ser servido como estático

**Conteúdo dinâmico (server Side)** **->** Gerado a partir de um código. Serviços que gera conteúdo de forma dinâmica (ex: consulta no banco de dados).

**Técnicas para reduzir quantidade de acesso ao banco de dados**

**HTML ->** Estrutura

**CSS** -> Estilo

**JS** -> comportamento

**DNS** -> Traduzir os nomes de domínio em endereços IP.

* Usuário informa o endereço no browser
* Computador envia uma consulta DNS para o servidor local
* Servidor responder com o endereço IP
* Computador acessa o servidor a partir do IP obtido

**Adquirir conhecimento básico sobre configuração de servidores**

**HTTPSeguro ->**

SSL4.0 = TLS

Criptografa todas as informações: URL, cookies e headers -> Só o ip é conhecido para que a comunicação possa acontecer.

LETS ENCRYPT é uma autoridade de certificação gratuita para certificar HTTPS

Se o navegador não tiver o certificado instalado, mesmo que o servidor tenha um certificado válido ele não será reconhecido(raro).

**Entender sobre certificado Digital e registro de HTTPS em CA**

**Web Service** -> provê interoperabilidade entre linguagens. Serviço de negócio que é disponibilizado em tecnologias para web. Os dados são trafegados de forma textual. Utiliza protocolo HTTP. xml com SOAP e JSON no REST (geralmente).

**SOA ->**  SOA é uma filosofia de TI que visa facilitar a integração entre sistemas, orientando a criação e a disponibilização de soluções modulares e fracamente acopladas baseadas no conceito de serviços.

**Microserviço não depende de software, é escalável, exige profissionais que conheçam e pratiquem a cultura DevOps**