Claude Code 토큰 효율 플레이북

효과적인 토큰 사용을 위한 기술, 워크플로우, 문화, 보안

사용량 급증 시대의 토큰 절감과 거버넌스

Choi, WooHyung

Prin. Solutions Architect / AWS Korea

whchoi@amazon.com

Agenda

0 비용의 두 축	단가 x 토큰 수 프레임, 폭증 원인
1 요금제와 단가	7개 플랜, 3-way 비교, 보조금, 전환, Bedrock
2 토큰 수 절감	기술 레버, MCP / Skills 로딩, 단가 레버
3 하네스 엔지니어링	컨텍스트 엔지니어링 포함, 롱러닝 핸드오프
4 캐시 - 두 축의 다리	캐시 경제성, 전환 비용을 좌우
5 Chat vs Co-Work	상세함도 토큰 구조에서 분기
6 PPT 워크플로우 분담	빌드 루프가 토큰의 가장 큰 소비
7 측정과 FinOps	무엇을 측정하나, 베이스라인
8 의사결정과 거버넌스	의사결정 규칙, 3-way 장단점, 보안
9 토큰 경제학과 사용 문화	사용량은 투자, 기본 경로가 다음 병목
10 보안과 거버넌스	안전한 모델 경로와 거버넌스 구현

00

비용의 두 축

단가 x 토큰 수

비용 = 단가 x 토큰 수

두 축을 캐시가 잇는다

프레임

비용은 단가 곱하기 토큰 수.

단가는 요금제 / 모델 / 캐싱 / 배치로, 토큰 수는 컨텍스트 / 하네스 / MCP / Skills로 줄인다. - 캐시가 두 축을 잇는 다리 역할.

단가 축 - 요금제 / 모델

어떤 플랜 (보조금 유무)
어떤 모델 (Opus / Sonnet / Haiku)
캐싱 / 배치 할인
= $ / 토큰을 정함

VS

토큰 수 축 - 효율

- 컨텍스트 관리 / 모델 믹스
- **하네스 엔지니어링**
- MCP / Skills 로딩 최적화

토큰 폭증의 원인

사용자 수만의 문제가 아니다

사용자

사용자 수 급증

채택 확산으로 동시 사용 증가.
좌석 / 플랜 압박.

세션

agentic 세션 규모

멀티스텝, 서브에이전트,
긴 컨텍스트로 세션당 토큰 폭증.

토크나이저

Opus 4.7+ 토크나이저

같은 텍스트에 최대 35%
더 많은 토큰.

한도

캡 한도 압박

주간 한도로 헤비 유저
throttle, 전환 검토 유발.

01

요금제와 단가

단가 축 - 보조금부터 전환까지

요금제 한눈에 - 7개 플랜

개인 / Team / Enterprise / Bedrock

플랜	가격 (2026.6)	과금 / 보이지 않는 보조금	성격
개인 Pro	~$17-20/월	정액, 보이지 않는 보조금 ~8.1x	라이트-중간
개인 Max 5x	$100/월	정액, 보이지 않는 보조금 ~13.5x	헤비 sweet spot
개인 Max 20x	$200/월	정액, 보이지 않는 보조금 13.5x (월 36.7x)	주간벽 상시 도달 시
Team Standard	$20-25/seat	정액, 세션 1.25x Pro	관리형 Pro+
Team Premium	$100-125/seat	정액, 세션 6.25x Pro	관리형 Max 5x 등가
Enterprise	~$20-60/seat + 실비	종량, 보이지 않는 보조금 없음	무상한 / 완전 거버넌스
Bedrock	Seat 비용 없음 + 실비	종량 , 보이지 않는 보조금 없음	프로그래매틱 / 자동화

참조 : https://she-llac.com/claude-limits / 추정치 이며 공식이 아님

개인 vs Enterprise vs Bedrock

같은 사용량, 다른 과금 구조

	개인 캡 플랜	Enterprise	AWS Bedrock 종량
과금 모델	정액 (상한 포함)	$20/seat + 토큰 실비	좌석 없이 토큰 실비
사용 한도	하드 캡 (5h + 주간)	무제한	무제한 (쿼터 관리)
숨은 보조금	약 8-37x	없음 (정가)	없음 (정가)
거버넌스	개인 없음 / Team SSO	SSO+SCIM, 감사, HIPAA	AWS IAM, CloudTrail, VPC
자동화	대화형 한정	대화형 중심	완전 프로그래매틱
적합	수십 석 미만 / 캡 내	무상한 / 컴플라이언스	AWS-native / 자동화

보조금의 정체 - credit 메커니즘

정액과 사용량의 분리

메커니즘

구독 한도는 token을 credit로 환산해 차감하고, credit 단가 비율은 API 달러 비율과 일치(입력:출력 1:5).

가격이 사용량과 분리된 정액이라 이 분리가 곧 보이지 않은 숨은 보조금.

환산 규칙

1. token to credit 로 차감
2. 입력 : 출력 = 1 : 5
3. Haiku : Sonnet : Opus = 1 : 3 : 5
4. 단가 비율 = API 달러 비율

정액 분리 효과

1. 가격이 사용량과 무관(정액)
2. 한도 소진 시 1M credit = $7.5 Opus
3. $200으로 $2,708어치 소비
4. = 13.5배 보조금 (추정)

1M credit
= $7.5 Opus

$200 → $2,708
Max 20x 가치(추정)

13.5x
보조금 배수(추정)

분리
보조금의 정체

개인 플랜 가치 배수 - Pro / Max 5x / Max 20x

두 Max는 가성비 동일

핵심

두 Max 가성비 동일(13.5x, 추정). 20x는 5x 대비 2배 비싼 2배 용량일 뿐 단위당 더 싸지 않다. 20x의 가치는 효율이 아니라 캐파 보험.

플랜	월 가격	월 credit (추정)	API 환산/월	배수(추정)
Pro	$20	21.7M	~$163	8.1x
Max 5x	$100	180.6M	~$1,354	13.5x
Max 20x	$200	361.1M	~$2,708	13.5x

참조 : https://she-llac.com/claude-limits / 추정치 이며 공식이 아님

Enterprise 전환 비용

숨은 보조금 소멸 = 실비로 전환

실비 전환시 비용 고려

Enterprise = $20/seat(접근권) + 토큰 API 정가, 무상한, 보조금 없음. 캐시 적중률에 따라 헤비 청구 16.8x-36.7x로 급증(아래는 추정).

케이스	현재	Enterprise 후 / 월	변화
솔로 라이트	Max 20x $200	$20 + ~$23 = $43	vs $200 저렴 (Pro $20보다는 비쌈)
솔로 웜 헤비	Max 20x $200	$20 + ~$7,335 = $7,355	약 36.7x ↑ (하한)
20인 팀 (개인)	Max 5x x 20 = $2,000	캡 (보조), 관리자 없음	-
20인 팀 (Team 혼합)	8 Prem + 12 Std = $1,040	캡 (보조) + SSO/미학습	개인 대비 ~$960/월 절약
20인 팀 (Enterprise)	$20x20 + 실비	~$28K-44K (월 ~$62.7K)	무제한, 실비

Bedrock 종량 - Seat 없는 실비 (AWS)

claude.ai UX가 불필요하면

Bedrock

claude.ai UX가 필요 없으면, 좌석 없이 동일 $5/$25 실비 + AWS-native 거버넌스 + Provisioned Throughput으로 프로그래매틱 멀티유저에 Enterprise보다 저렴할 수 있다. (인리전 엔드포인트는 +10%.)

과금	거버넌스	예측성	정산
좌석 없이 실비	AWS-native	Provisioned Throughput	per-user 귀속
$5/$25 정가, 시트료 0. 캐시 + Batch 적용.	IAM, CloudTrail, VPC, 데이터 레지던시.	용량 예약으로 비용 / 지연 예측.	per-user IAM Role + CloudTrail 기반 정산.

02

토큰 수 절감

토큰 수 축 - 기술 레버

기술 절감 레버 종합

세션 단위에서 줄이는 네 가지

컨텍스트

컨텍스트 관리

불필요 히스토리 제거, compaction, 고신호 토큰만 유지.

모델

모델 믹스

Sonnet 기본, Opus 복잡 10%, Haiku 분류 / 라우팅.

도구

MCP / Skills 로딩

필요할 때만 로드. on-demand로 상시 점유 최소화.

규칙

CLAUDE.md / Hooks

지침은 짧게, 결정적 작업은 스크립트 / 훅으로.

컨텍스트 관리

토큰 절감 효과가 가장 큰 영역

원리

토큰 비용은 컨텍스트 크기에 비례. prompt caching과 auto-compaction은 기본 자동 최적화.

나머지는 컨텍스트를 작게 유지하는 습관.

/clear

작업 전환 초기화

무관한 작업 전환 시.
/rename 보관 후
/resume 복귀.

/compact

대화 요약 압축

보존 지시 동봉.
CLAUDE.md에 compact
지침 명시.

/usage

사용량 가시화

skill / subagent / plugin
/ MCP별 비중표시 , d / w
토글 사용.

/context

토큰 분해 확인

시스템 / 툴 / 메모리 / 스킬
/ 대화별 점유.

기술적 절감 2 - CLAUDE.md & Skills

세션마다 로드되는 컨텍스트를 가볍게

원리

CLAUDE.md는 세션 시작마다 통째로 로드 - 짧을수록 매 세션 절감.

워크플로우 지식은 필요할 때만 부르는 Skill로 분리(progressive disclosure).

CLAUDE.md

린하게 유지

200줄 이하.
매 세션 필요한 핵심만, 장황한 규칙은 곧 비용.

Skills

on-demand 로딩

PR 리뷰 / DB 마이그레이션 등 특정 워크플로우 지침을 분리.

progressive

필요할 때만 주입

도메인 지식을 상시 적재하지 않고 호출 시점에.

자동 최적화

기본 동작

prompt caching + auto-compaction이 백그라운드로.

MCP 로딩과 토큰 점유

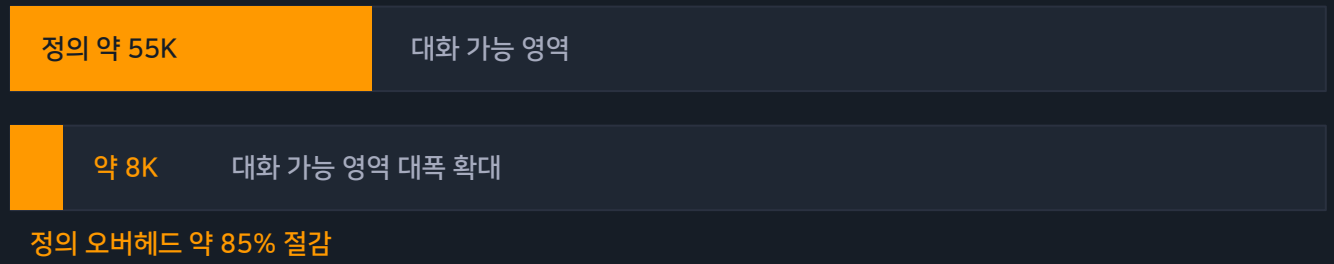
도구 정의가 대화 전부터 컨텍스트를 차지

이해

MCP는 연결 즉시 모든 도구 정의 + 도구용 시스템 프롬프트를 컨텍스트에 적재.

정의 / 호출 / 결과가 모두 input 토큰. 정의당 50-500토큰, 시스템 프롬프트 약 290-590토큰.

200K 컨텍스트 기준 예시 - 도구 정의가 차지하는 비중



모범 사례

미사용 서버 /mcp 비활성, on-demand 도구 검색으로 정의 오버헤드 약 85% 절감, 단순 작업은 CLI(gh / aws / gcloud)로 대체.

Skills 단계 로딩 (progressive disclosure)

필요한 만큼만 컨텍스트에

Skills

Skills는 3단계로 로드된다. metadata는 상시(가볍게), SKILL.md 본문은 작업 매칭 시, 번들 스크립트는 실행 시(컨텍스트에 안 올림).
정의는 정밀하게, 본문은 짧게.

한 번에 다 로드 (비효율)

모든 지침 + 예시 상주
쓰지 않아도 토큰 점유
스킬 늘수록 선형 증가
= 낭비

VS

단계 로딩 (효율)

- metadata 상시 약 100 토큰 / 스킬
- SKILL.md 매칭 시 약 5K
- 스크립트 실행 시 0 (미적재)

모델과 추론 노력

비싼 건 모델이 아니라 잘못된 기본값

기본 전략

Sonnet 기본 / Opus는 복잡한 추론에만 / 단순 subagent는 haiku.

Plan은 Opus, 실행은 Sonnet 자동 전환이 opusplan. (서드파티 배포 별칭 / 핀은 다음 장)

모델 선택

Sonnet 기본

Opus는 복잡한 추론 한정, 단순 subagent는 haiku.

opusplan

Opus→Sonnet 자동

계획은 Opus, 실행은 Sonnet 자동 전환.

thinking 절감

effort 낮춤

/effort, MAX_THINKING_TOKENS=8000.

Fable 5는 항상 thinking.

전환과 캐시

수동 전환은 비용

이전 출력 있는 세션서 모델 변경 시 캐시 없이 전체 재처리.

opusplan 자동이 친화적.

Sonnet
일상 기본 모델

opusplan
계획 / 실행 자동

8000
MAX_THINKING_TOKENS

전환=무효화
모델 변경 시 캐시

기타 레버

MCP, Hooks, Subagents, 프롬프트 규율

프롬프트 규율

구체적 프롬프트 + plan mode(Shift+Tab) + 검증 타깃 + 점진적 테스트. 어긋나면 Esc, /rewind로 체크포인트 복귀.

MCP / 툴

정의 deferred

이름만 로드.
미사용 /mcp 비활성,
CLI(gh/aws/gcloud) 우선.

Hooks 전처리

보기 전에 가공

PreToolUse로 1만 줄 로그를
ERROR만 - 수백 토큰으로.

Subagents

verbose 격리

테스트 / 로그 / fetch는 격
리, 요약만 메인으로 회수.

Agent teams

약 7배 토큰

plan mode 시 독립 컨텍스트.
작게, 끝나면 종료. 기본 비활성.

단가 레버 종합

토큰 수와 별개로 단가를 낮춘다

단가 레버

모델 선택 / 캐싱 / 배치 / 출력 길이가 토큰당 단가를 좌우.

출력이 입력의 5배라 출력 길이 통제가 가장 큰 레버.

레버	효과	비고
모델 믹스	Haiku 1 / Sonnet 3 / Opus 5 ($/MTok in)	출력은 5배. 5-25x 단가 격차
프롬프트 캐싱	cache read 0.1x (-90%)	write 5분 1.25x, 1시간 2x
Batch API	입출력 -50%	비동기 24h, 품질 동일
출력 길이	출력 = 입력의 5배 단가	가장 큰 비용 드라이버
라우팅 / 리전	US-only 1.1x (인리전), Bedrock 리전 +10%	기본 global이 저렴

03

하네스 엔지니어링

컨텍스트 엔지니어링의 상위 프레임

하네스 엔지니어링이란

context engineering을 포함하는 더 넓은 설계

정의

하네스 = 모델을 감싸는 스캐폴딩 - 멀티스텝 루프 / 툴 중재 / 검증 / 지속 상태를 관리.

context engineering(한 호출의 컨텍스트 설계)을 포함. 원칙: 유한한 어텐션 예산에서 최소 고신호 토큰.

하네스 엔지니어링 (에이전트 스캐폴딩)

컨텍스트 엔지니어링
한 호출의 컨텍스트 설계
최소 고신호 토큰

- 멀티스텝 루프 - reason / act / reflection
- 툴 중재 - MCP / Skills / CLI 선택
- 검증 게이트 - 평가자 분리로 탈선 차단
- 지속 상태 - 아티팩트 / compaction

롱러닝 하네스와 토큰 비용

컨텍스트를 윈도우 너머로 넘긴다

롱러닝

긴 작업은 단일 컨텍스트 윈도우를 넘는다.

아티팩트(feature-list / progress-log / git)로 상태를 외부화해 윈도우 간 핸드오프하면 재로딩 / 재작업 토큰을 줄인다.

단일 윈도우에 다 욱여넣기

컨텍스트 한계 도달
재로딩 / 재작업 반복
누적 토큰 폭증
= 비효율 + 품질 저하

VS

아티팩트 핸드오프 (효율)

- feature-list / progress-log
- **git으로 상태 외부화**
- Agent SDK 자동 compaction

04

캐시 - 두 축의 브릿지

단가와 토큰 수를 함께 좌우

캐시 경제성

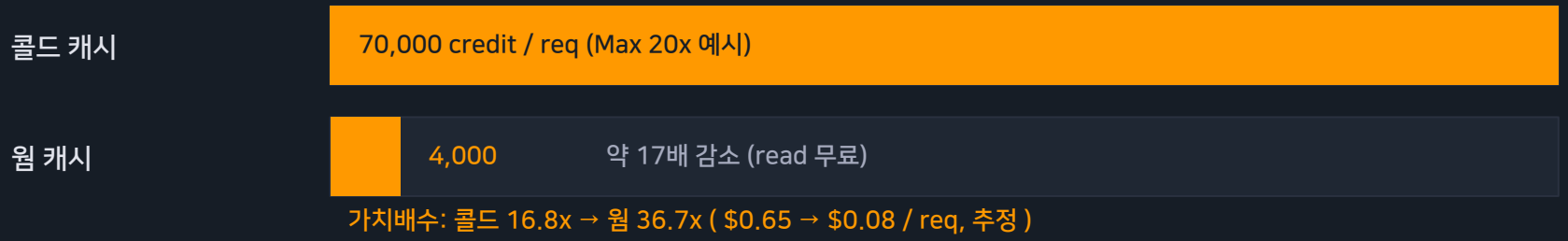
구독은 무료, 정가는 0.1x

캐시

구독은 cache read 0 credit(무료), write 1x. 정가(API)는 read 0.1x(-90%), write 5분 1.25x / 1시간 2x.

같은 컨텍스트를 반복 read하는 agentic 루프일수록 작업당 비용 급감.

구독 - 캐시 상태별 req당 credit (100K 컨텍스트 예시)



캐시가 좌우하는 전환 비용

효율이 곧 청구서 절감

연결

같은 사용량도 캐시 적중률이 청구를 좌우. 캡에서는 효율이 한도 여유를, 종량에서는 효율이 직접 비용 절감을 만든다.

전환 검토 시 효율을 함께 적용해 실비를 낮춰야 한다.

캡 플랜 (구독)

1. cache read 0 credit (무료)
2. 웜일수록 보조금 16.8x → 36.7x
3. 정액이라 청구 변동 없음

효율 = 한도 여유

종량 (Enterprise / Bedrock)

1. cache read 0.1x, write 1.25-2x
2. 웜 전환 비용 16.8x → 36.7x 격차
3. 토큰 수 x 캐시 = 실제 청구
4. 효율 = 직접 비용 절감

05

Chat vs Cowork

상세함도 토큰도 구조에서 갈립니다

Chat vs Cowork - 구조 차이

상세함도 토큰도 구조에서 갈립니다

왜 토큰을 더 쓰나

에이전트 루프는 매 turn 전체 대화와 도구 결과를 다시 입력 - 단계가 늘수록 입력이 누적.

시스템 프롬프트와 툴 정의의 고정 오버헤드도 Chat보다 큼.

Chat (단발)

1. 검색 1회 후 바로 응답
2. 도구 사용 제한적
3. 결과가 짧고 단순
4. 고정 오버헤드 작음

Cowork (에이전트)

1. 재검색 / 코드실행 / 교차확인 반복
2. 파일 생성 전제, 산출물 지향
3. 결과가 길고 구조화
4. 검증 보강 루프가 기본

Prompt caching - 비용과 한계

누적이 비용으로 선형 증가하진 않습니다

수치

cache read = base input의 10% (0.1x).

write = 5분 1.25x / 1시간 2x. 5분 캐시는 1회 읽으면 본전, 멀티턴 입력은 약 5-10배 절감.

한계 1

prefix 캐시

접두부(시스템 / 툴 정의)가 1토큰만 달라도 전체 miss.
새 툴 추가는 전 프롬프트 무효화.

한계 2

출력은 캐시 안 됨

출력 토큰은 항상 full price.
생성 중심 작업일수록 절감 헤드라인과 멀어짐.

한계 3

전환 = 무효화

모델 전환 시 다음 응답이 캐시 없이 전체 히스토리 재처리.

06

PPT 워크플로우 분담

빌드 루프가 토큰을 가장 많이 태웁니다

PPT 워크플로우 분담 - 어디서 무엇을

빌드 루프가 토큰을 가장 많이 태웁니다

핵심

빌드 루프(코드 → PDF / JPG QA → 수정 → 재빌드)는 출력 중심이라 캐시로 못 줄임.

유일한 레버는 루프 횟수 줄이기 = 명세 선확정.

Chat에서 확정 (싼 단계)

1. 목차, 장표 수
2. 각 장 핵심 메시지와 본문
3. 청중 규모, 발표자 정보
4. 강조 수치, 다이어그램 구조

Cowork로 넘김 (도구 실행)

1. 스킬 적용, OOXML 후처리
2. PDF / JPG 시각 QA
3. 확정 명세로 빌드, 수정만
4. 콘텐츠 재고민은 금물

분담 결정 규칙과 손익분기점

항상 Chat 선행이 이득은 아닙니다

상황별 권장	
5-10장, 콘텐츠 명확	Cowork 직행
중대형 / 콘텐츠 유동적 / QA 반복 예상	Chat 명세 확정 후 Cowork 빌드
워크숍, 시리즈 (수백 장)	Chat 선행 필수

실무 팁

1. 명세를 Cowork 첫 메시지에 통째로 붙여넣어 되묻기 turn을 줄임.
2. QA는 전 슬라이드를 모아 1회 사이클로.
3. 손익분기점은 A / B (Cowork 단독 vs Chat 후 Cowork) 누적 토큰 비교로 확정.

07

측정과 FinOps

두 레버를 분리해 추적

측정과 가시성

효율은 가르칠 수 있는 습관입니다

출발점

같은 코드베이스, 같은 CLAUDE.md라도 사용자와 프롬프트 구조에 따라 cache 효율이 크게 갈림(사례 기반).

측정 없이는 누가 잘하는지 보이지 않음.

OTel 텔레메트리

토큰 / 비용 메트릭

token.usage(cacheRead 포함),
cost.usage(model/agent/skill).
prompt.id 상관.

managed settings

조직 강제

IT가 전체에 적용, 사용자 override 불가.
MDM / Admin Console.

Analytics

API / 대시보드

팀별 비용 배분, 생산성.
contribution은 beta, ZDR은 usage만.

Bedrock 비용

LiteLLM 추적

Claude Code가 클라우드 비용 미전송 - key별 spend로.

cache hit %
팀 KPI 후보

override 불가
managed settings

ZDR = usage만
contribution 제한

LiteLLM
Bedrock 비용 추적

무엇을 측정하나

캐시 적중률이 단일 지배 변수

토큰

토큰 / 세션

세션당 입출력 토큰, 캐시 read / write 비중.

캐시

캐시 적중률

cache read 비율.
비용의 단일 지배 변수.

모델

모델 믹스

Opus / Sonnet / Haiku 비율.
단가 분포.

비용

$ / 작업

작업당 비용, 베이스라인 대비 추세.

베이스라인과 표준

FinOps 케이던스로 습관을 정착

베이스라인

파일럿 후 확대

평균 $13 / active day, 월 $150-250, 90%가 $30 / day 미만 등 예상 이상치 기준선 수립 (예시)

Rate limit

규모별 TPM / RPM

200명이면 1인 20k, 총 4M. 워크스페이스 spend limit.

공유 표준

팀 자산화

팀 CLAUDE.md, 사내 Skill 라이브러리, managed settings 기본값.

교육 / 프레이밍

습관과 신뢰

온보딩 체크리스트, 챔피언. cost 아닌 quality로, DORA 병행.

$13 / day	90% < $30	TPM / RPM	DORA 병행
dev당 평균 (active)	active day 비용	규모별 권장	usage 는 outcome 아님

베이스라인과 FinOps

단가 레버와 토큰 수 레버를 나눠서

FinOps

단가 레버와 토큰 수 레버를 분리해 추적. OTEL 텔레메트리, Enterprise Analytics API, 대시보드로 베이스라인 대비 추세를 본다.

단가 레버 KPI

1. 모델 믹스 비율
2. 캐시 적중률
3. 배치 사용률
4. global 라우팅 비율

토큰 수 레버 KPI

1. 세션당 토큰
2. MCP / Skills 점유
3. 재작업 / 재로딩 비율
4. 출력 길이 분포

08

의사결정과 거버넌스

무엇을 고르고 어떻게 통제할까

의사결정 규칙

단가 선택 + 효율을 함께

상황	권고
5석 미만, 대화형 중심	개인 플랜. Max 5x sweet spot (주간벽 상시 도달 시에만 20x)
5-150석, 캡 안에서 충분	Team(혼합). 보조금 유지 + 거버넌스
무상한 / 150석 초과 / SCIM,감사,HIPAA	Enterprise (정가 실비, throttle 소멸)
프로그래매틱 / 자동화 / 레지던시 / AWS	Bedrock 종량 (좌석료 0 + AWS 통제)
어느 경로든 공통	캐시 적중률 + 토큰 수 절감 + 모델 믹스를 먼저

장단점 종합 - 3-way

각 경로의 득과 실

요약

캡은 보조금과 캡, Enterprise는 무상한과 정가, Bedrock은 실비와 자동화. 헤비일수록 종량 충격이 크다.

개인 / Team

보조금 / 캡

+ 8-37x 보조금, 정액, cache read 무료.
- 하드 캡 / throttle, 불투명, 대화형 한정.

Enterprise

무상한 / 정가

+ throttle 소멸, 완전 거버넌스.
- 보조금 전무(헤비 10-37x+), 지출 변동성.

Bedrock

실비 / 자동화

+ 좌석료 0, AWS-native, 프로그래매틱.
- UX 제약, 운영 부담, 정가.

거버넌스 / 보안 요점

비용 통제는 곧 거버넌스

비용

spend cap

Enterprise admin 지출 상한으로 예산 통제.

데이터

미학습 / 레지던시

Team / Enterprise 기본 미학습 Bedrock 데이터 레지던시.

접근

IAM / SSO+SCIM

per-user IAM(Bedrock), SSO+SCIM(Enterprise).

감사

CloudTrail / 감사로그

호출 추적, 사용자별 귀속, 컴플라이언스.

09

토큰 경제학과 사용 문화

비용은 사용량이 아니라 기본 경로에서

비용은 사용량이 아니라 설계

사용량은 키우고, 단가는 잡는다

관점

비용 = 사용량 x 단가. 건강한 조직은 사용량을 의도적으로 키우고(투자), 비용은 단가에서 잡는다.

사용량을 누르지 않고 기본 경로를 설계한다.

사용량을 누르는 통제 (잘못)

한도 ↓ / 경고 / 승인제
AI를 덜 쓰게 됨
비용은 그대로
문화만 죽는다

VS

경로를 설계 (올바름)

- 사용량 ↑ = 투자로 본다
- **단가를 설계로 낮춘다**
- 기본 경로를 다시 짠다

흔한 오해 - 과다 사용자가 아니다

비용은 기본 경로에서 나온다

오해

비용은 소수의 과다 사용자에서 나오지 않는다. 대다수는 한도 근처에도 안 간다.

비용은 조직 전체의 기본 경로가 비싸게 설계된 데서 나온다.

오해

몇몇이 너무 많이 쓴다

소수 과다 사용자가 비용을 만든다는 통념.

사실

대다수는 한도 미달

직원 대다수는 사용 한도 근처에도 가지 않는다.

진짜 원인

비싼 기본 경로

한도 / 경고 / 승인제는 사용량만 누른다. 가시성은 통제가 아니라 설계 수단.

단가의 세 지렛대

가장 강력한 건 안 만드는 것

지렛대

단가를 낮추는 세 지렛대.

가장 강력한 건 처음부터 만들지 않는 것 - 안 만든 토큰의 비용은 0.

1 (가장 강력)

처음부터 안 만들기

1. Skill로 검증된 방식(규칙 / 구조 / 제약)을 미리 담아 첫 결과를 정답에 가깝게 → 반복 / 리뷰 감소.
2. Plugin으로 업무 흐름을 묶는다.

2

재사용

1. 같은 맥락 / 요청 반복이면 매번 새로 생성하지 않고 결과를 재사용
   → 단가↓ (캐싱).

3

모델 분배

1. Opus 4.8 어려운 병목, Sonnet 4.6 기본, Haiku 4.5 분류 / 대량 / 1차 초안.
2. Opus는 병목에만.

문화 - 더 쓰는 문화를 먼저

질문의 방향을 바꾼다

순서

단가 최적화부터 들이밀면 위축돼 안 쓴다.

더 쓰는 문화를 먼저, 그 위에 단가 설계. 질문을 바꾼다 - 토큰을 어떻게 아낄까(X) → 오늘 무엇을 더 맡길까(O).

더 쓰는 문화 (먼저)

1. 질문 전환 - 무엇을 더 맡길까
2. 실제 업무 흐름 안에 붙이기
3. 잘 쓰는 사용자 분석 / 시연 / 확산
4. 반복 업무 자동화로 확산

그 위에 단가 (나중)

1. 모델 분배 / 재사용 / 안 만들기
2. 측정으로 비싼 경로 진단
3. 경로 변경 효과를 숫자로 검증
4. 위축 없이 비용 ↓

Skills / Plugins 는 공유 자산

개인 도구가 아니라 조직 표준으로

공유

Skill / Plugin을 개인 임시 도구로 두면 효과는 한 사람에서 끝난다.

조직 공유 자산으로 다뤄야 한다. 확산에 쓴 방법을 낭비 절감에도 똑같이 적용.

개인 임시 도구 (효과 국소)

그 사람 안에서 끝남

표준 없음

중복 재발명

전사 효과 없음

VS

조직 공유 자산 (전사 확산)

잘 쓰는 사람의 것 발견

표준으로 정리

중앙 저장소로 배포

결국 하나의 문제 - 기본 경로

경제와 문화가 만나는 곳

종합

좋은 기본 경로 = 의식 않고도 많이 쓰고(문화), 그 사용이 더 싸고 나은 경로로 흐른다(경제).

비용 관리는 중앙 억제가 아니라 플랫폼 설계. 다음 병목은 모델 성능도 토큰 비용도 아닌 기본 경로.

문화

많이 쓴다

사용자가 의식하지 않고도 계속 많이 쓴다.

경제

싸게 흐른다

그 사용이 더 싸고 나은 경로로 자연스럽게.

설계

플랫폼 문제

가시성 / 모델 분배 / 자동 분배 / 재사용 / 공유 Skill / 맥락 최소 / ROI 기준.

병목

기본 경로

다음 병목은 성능도 비용도 아닌, 조직이 정해 둔 기본 경로.

10

보안과 거버넌스

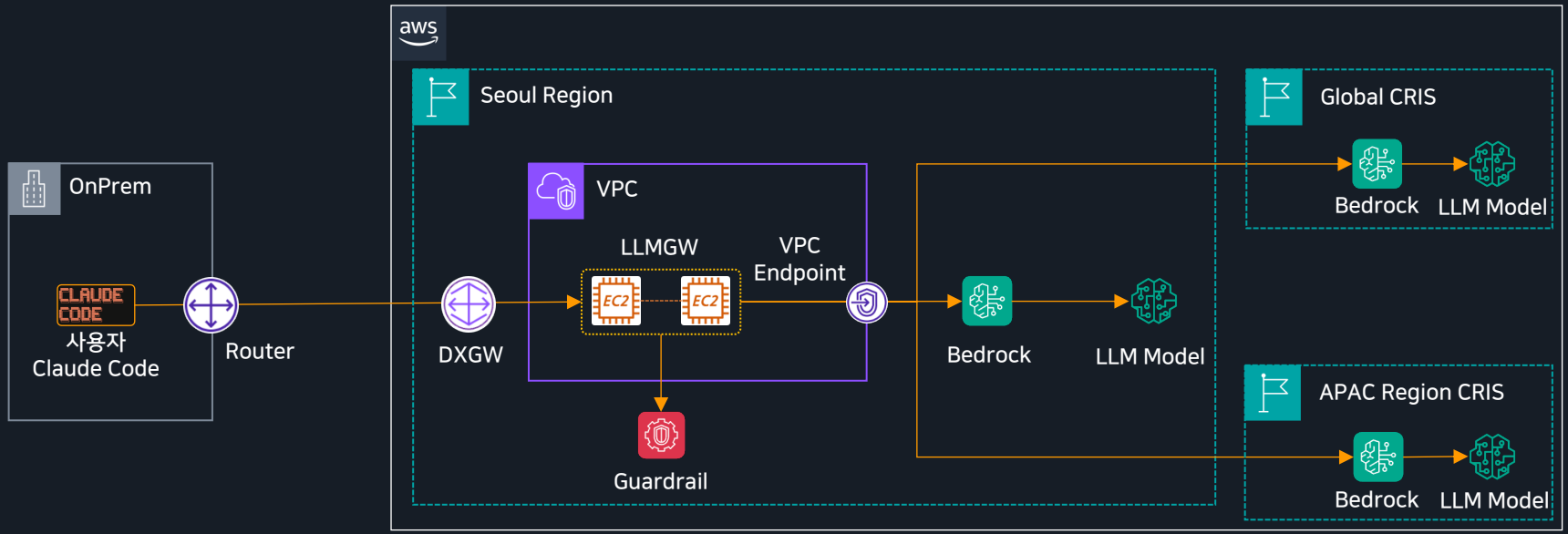
안전한 모델경로와 거버넌스 구현

보안과 거버넌스를 위한 아키텍쳐

안전한 모델 사용 경로를 제공

안전한 모델 사용 경로를 제공

온프렘 사용자들을 위한 안전한 모델 사용 경로를 제공 – InRegion , Region CRIS, Global CRIS



보안과 거버넌스 1 - 접근과 통제

에이전트는 실제 동작을 수행합니다

전제
Claude Code는 파일 편집, 명령 실행, 도구 호출을 자율 수행.
통제는 IT가 강제하는 managed settings를 baseline으로 둠.

managed settings

조직 보안 기준선

IT 배포(MDM: Jamf / Intune, Admin Console). 사용자 override 불가.

permission 통제

사전 검토 게이트

plan mode 검토, allowed - denied 도구, 위험 동작 승인.

Hooks = control

동작 차단 코드

민감 동작 차단.
훅 등록 / 실행 / 실패 / 차단 결정 모니터링.

MCP 신뢰 경계

주입 위험 관리

미사용 비활성, CLI 우선.
도구 결과 / 웹기반 prompt injection 주의.

보안과 거버넌스 2 - 데이터와 관측

무엇이 로깅, 캐시되는지 통제

데이터 경계

prompt caching은 ZDR 적격 - 원문 미저장, KV 캐시와 해시는 메모리에만, 조직(및 워크스페이스)간 격리.

redaction

민감정보 마스킹

프롬프트 내용 로깅, 도구 details - user.email은 collector에서 redaction.

opt-in 원칙

기본 미전송

내보내기는 명시 설정 시에만. 기본은 아무 데이터도 흐르지 않음.

OTel → SIEM

보안 모니터링

Splunk / Datadog / Sentinel / Elastic. 권한 결정까지 상관.

감사

컴플라이언스

Compliance API로 감사. Cowork는 공유 식별자로 상관.

효율과 보안 - 같은 레버, 두 목적

토큰을 줄이는 지점이 곧 통제점

레버	효율 측면	보안 측면
managed settings	모델 / 텔레메트리 기본값 표준화	override 불가 정책 강제, 기준선
텔레메트리(OTel)	cache hit / 비용 추적	권한 결정 / 이상 행위 SIEM
Hooks	로그 전처리로 토큰 절감	민감 동작 차단 control code
MCP 정리	미사용 정의 제거로 오버헤드 감소	공격면 축소, injection 관리
prompt caching	입력 재처리 비용 할인	ZDR 적격, 조직 간 격리

부록 - 치트시트와 캐싱 비용

현장에서 바로 쓰는 레퍼런스

명령어 치트시트	
초기화	/clear, /rename, /resume
압축	/compact <보존 지시>
사용량	/usage (d / w), /context
모델	/model, opusplan, haiku
thinking	/effort, MAX_THINKING_TOKENS
MCP	/mcp (미사용 비활성)
되돌리기	Esc, /rewind

caching 비용 (base input 대비)	
cache read	0.1x (10%)
write 5분	1.25x
write 1시간	2x
출력 토큰	캐시 불가, full price
모델 prefix 변경	전체 miss

부록 - 보안 / 거버넌스

배포 전 점검 레퍼런스

접근과 통제

정책 강제	managed settings (override 불가)
권한	plan mode, allowed / denied
차단	Hooks (PreToolUse) + 훅 모니터링
신뢰 경계	/mcp 비활성, CLI 우선

데이터와 관측

데이터 경계	prompt caching (ZDR 적격)
로그 가림	OTEL_LOG_* redaction
내보내기	OTLP exporter (opt-in)
모니터링	OTel - Splunk / Datadog / Sentinel
감사	Compliance API

Thank you!

Choi, WooHyung / Prin. Solutions Architect / AWS Korea
whchoi@amazon.com / linkedin.com/in/woohyungchoi